50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. DOD/SW/MT-88/010 a	2.

AD-A188 542

4. Title and Subtitle
Version Description Document for the Ada Compiler Validation Capability (Version 1.10): ANSI

6.

7. Author(s)

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
ASD/SCOL
Language Control Division
Area B, Bldg 676
Wright-Patterson AFB, OH 45433-6503

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C)
(G)

12. Sponsoring Organization Name and Address

13. Type of Report & Period Covered

14.

15. Supplementary Notes
For magnetic tape, see:

16. Abstract (Limit: 200 words)

The Ada Compiler Validation Capability (ACVC), Version 1.10, consists of 3719 tests in 4182 test files. Also, there are three units in four files and three tests in three files which support the running of the ACVC tests; one data file containing macro definitions; one program in one file for making the macro substitutions; and five programs in 165 files which are used to aid in the validation process. The differences between Version 1.9 and 1.10 are detailed in sections 2.0 through 4.0.

17. Document Analysis a. Descriptors
The ACVC is a suite of tests designed to test an Ada Compiler's Conformance to MIL-STD-1815A.

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
	20. Security Class (This Page)	22. Price

(See ANSI-Z39.18) See Instructions on Reverse OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce

VERSION DESCRIPTION DOCUMENT
FOR THE
Ada COMPILER VALIDATION CAPABILITY
(Version 1.10)

1 VERSION IDENTIFICATION

This document describes the Ada Compiler Validation Capability (ACVC), Version 1.10. This version consists of 3719 tests in 4182 test files. Also, there are three units in four files and three tests in three files which support the running of the ACVC tests; one data file containing macro definitions; one program in one file for making the macro substitutions; and five programs in 165 files which are used to aid in the validation process. The differences between Version 1.9 and 1.10 are detailed in sections 2.0 through 4.0.

2 TEST MODIFICATIONS

In Version 1.10 of the ACVC, 383 tests were modified from the corresponding tests in Version 1.9. Modifications have been made:

a) to correct tests that were found to be incorrect in previous versions of the ACVC,
b) to strengthen the tests by making the tests more comprehensive or by incorporating more cases to check, and
c) to bring test code into conformance with the coding standards.

The following tests have been modified from Version 1.9:

A28004A.ADA	A35902C.ADA	A74106C.ADA	AD1A01A.ADA
AE3101A.ADA	B22003A.ADA	B23003F.TST	B24204A.ADA
B24204B.ADA	B24204C.ADA	B28001J.ADA	B28001R.ADA
B28001S.ADA	B28001T.ADA	B28003A.ADA	B2A003A.ADA
B2A003B.ADA	B2A003C.ADA	B33301A.ADA	B36001A.ADA
B37201A.ADA	B37312B.ADA	B39004G.ADA	B39004J.ADA
B39004L.ADA	B44001A.ADA	B49006A.ADA	B49009A.ADA
B59001A.ADA	B64001A.ADA	B67001A.ADA	B67001B.ADA
B67001C.ADA	B67001D.ADA	B85013C.ADA	B86001B.ADA
B91001H.ADA	B91003B.ADA	B95001A.ADA	B97101A.ADA
B97101E.ADA	B97102C.ADA	B97102E.ADA	BC3105A.ADA
BC3604A.ADA	C32001B.ADA	C32111A.ADA	C32112A.ADA
C32112B.ADA	C32113A.ADA	C32115A.ADA	C34004A.ADA
C34007A.ADA	C34007D.ADA	C34007G.ADA	C34007M.ADA
C34007P.ADA	C34007S.ADA	C34012A.ADA	C35102A.ADA
C35502F.TST	C35502K.ADA	C35502N.DEP	C35502P.ADA
C35503C.ADA	C35503D.TST	C35503E.ADA	C35503F.TST
C35503G.ADA	C35503H.ADA	C35503K.ADA	C35503L.ADA
C35503O.ADA	C35503P.ADA	C35505C.ADA	C35507C.ADA
C35507E.ADA	C35507G.ADA	C35507H.ADA	C35507I.DEP

C35507J.DEP	C35507K.ADA	C35507M.DEP	C35507N.DEP
C35508E.ADA	C35508G.ADA	C35508H.ADA	C35508J.DEP
C35508O.ADA	C35508P.ADA	C35703A.ADA	C35711B.ADA
C35802Y.DEP	C35802Z.DEP	C35904A.ADA	C35904B.ADA
C35A06N.ADA	C36204B.ADA	C36204C.ADA	C37107A.ADA
C37206A.ADA	C37207A.ADA	C37209B.ADA	C37213H.ADA
C37213J.ADA	C37215C.ADA	C37215E.ADA	C37215G.ADA
C37215H.ADA	C38102C.ADA	C39005A.ADA	C41307D.ADA
C41308C.ADA	C41320A.ADA	C41402A.ADA	C42007F.ADA
C42007G.ADA	C45242B.ADA	C45252B.ADA	C45273A.ADA
C45332A.ADA	C45532B.DEP	C45532F.DEP	C45532J.DEP
C45532N.DEP	C45614C.DEP	C45621G.DEP	C45621H.DEP
C45621I.DEP	C45621J.DEP	C45621K.DEP	C45621L.DEP
C45621M.DEP	C45621N.DEP	C45621O.DEP	C45621P.DEP
C45621Q.DEP	C45621R.DEP	C45621S.DEP	C45621T.DEP
C45621U.DEP	C45621V.DEP	C45621W.DEP	C45621X.DEP
C45621Y.DEP	C45621Z.DEP	C46014A.ADA	C47005A.ADA
C47008A.ADA	C47009A.ADA	C47009B.ADA	C4A012B.ADA
C52104L.ADA	C54A24B.ADA	C64103B.ADA	C64103E.ADA
C64103F.ADA	C64104A.ADA	C64104H.ADA	C64104I.ADA
C64104J.ADA	C64104K.ADA	C64104O.ADA	C64109H.ADA
C64109I.ADA	C64109J.ADA	C64109K.ADA	C64109L.ADA
C85018B.ADA	C87B04B.ADA	C87B06A.ADA	C87B19A.ADA
C94002G.ADA	C96001A.ADA	C97301B.ADA	C99004A.ADA
CC1221A.ADA	CC1311A.ADA	CC1311B.ADA	CC3603A.ADA
CE2102A.ADA	CE2102B.ADA	CE2102C.TST	CE2102D.ADA
CE2102E.ADA	CE2102F.ADA	CE2102G.ADA	CE2102H.TST
CE2102I.ADA	CE2102J.ADA	CE2102K.ADA	CE2103A.TST
CE2103B.TST	CE2104A.ADA	CE2104B.ADA	CE2104C.ADA
CE2104D.ADA	CE2105A.ADA	CE2105B.ADA	CE2106A.ADA
CE2106B.ADA	CE2107A.ADA	CE2107B.ADA	CE2107C.ADA
CE2107D.ADA	CE2107E.ADA	CE2107F.ADA	CE2107G.ADA
CE2107H.ADA	CE2107I.ADA	CE2108A.ADA	CE2108B.ADA
CE2108C.ADA	CE2108D.ADA	CE2109A.ADA	CE2109B.ADA
CE2109C.ADA	CE2110A.ADA	CE2110B.ADA	CE2110C.ADA
CE2111A.ADA	CE2111B.ADA	CE2111C.ADA	CE2111D.ADA
CE2111E.ADA	CE2111G.ADA	CE2111H.ADA	CE2115A.ADA
CE2115B.ADA	CE2201A.ADA	CE2201B.ADA	CE2201C.ADA
CE2201F.ADA	CE2201G.ADA	CE2202A.ADA	CE2204A.ADA
CE2208B.ADA	CE2401A.ADA	CE2401B.ADA	CE2401C.ADA
CE2401E.ADA	CE2401F.ADA	CE2401H.ADA	CE2402A.ADA
CE2404A.ADA	CE2405B.ADA	CE2406A.ADA	CE2407A.ADA
CE2408A.ADA	CE2409A.ADA	CE2410A.ADA	CE2411A.ADA
CE3002B.TST	CE3002C.TST	CE3102A.ADA	CE3102B.TST
CE3103A.ADA	CE3104A.ADA	CE3107A.TST	CE3108A.ADA
CE3108B.ADA	CE3109A.ADA	CE3110A.ADA	CE3111A.ADA
CE3111B.ADA	CE3111C.ADA	CE3111D.ADA	CE3111E.ADA
CE3112A.ADA	CE3112B.ADA	CE3114A.ADA	CE3114B.ADA
CE3115A.ADA	CE3206A.ADA	CE3208A.ADA	CE3301A.ADA

CE3302A.ADA	CE3303A.ADA	CE3305A.ADA	CE3402A.ADA
CE3402C.ADA	CE3402D.ADA	CE3402E.ADA	CE3403A.ADA
CE3403B.ADA	CE3403C.ADA	CE3403D.ADA	CE3403E.ADA
CE3403F.ADA	CE3404A.ADA	CE3404B.ADA	CE3404C.ADA
CE3405A.ADA	CE3405C.ADA	CE3405D.ADA	CE3406A.ADA
CE3406B.ADA	CE3406C.ADA	CE3406D.ADA	CE3407A.ADA
CE3407B.ADA	CE3407C.ADA	CE3408A.ADA	CE3408B.ADA
CE3408C.ADA	CE3409A.ADA	CE3409B.ADA	CE3409C.ADA
CE3409D.ADA	CE3409E.ADA	CE3410A.ADA	CE3410B.ADA
CE3410C.ADA	CE3410D.ADA	CE3410E.ADA	CE3411A.ADA
CE3411C.ADA	CE3412A.ADA	CE3413A.ADA	CE3413C.ADA
CE3601A.ADA	CE3602A.ADA	CE3602B.ADA	CE3602C.ADA
CE3602D.ADA	CE3603A.ADA	CE3604A.ADA	CE3605A.ADA
CE3605B.ADA	CE3605C.ADA	CE3605D.ADA	CE3605E.ADA
CE3606A.ADA	CE3606B.ADA	CE3701A.ADA	CE3704A.ADA
CE3704B.ADA	CE3704C.ADA	CE3704D.ADA	CE3704E.ADA
CE3704F.ADA	CE3704M.ADA	CE3704N.ADA	CE3704O.ADA
CE3706C.ADA	CE3706D.ADA	CE3706F.ADA	CE3706G.ADA
CE3707A.ADA	CE3708A.ADA	CE3801A.ADA	CE3804A.ADA
CE3804B.ADA	CE3804C.ADA	CE3804D.ADA	CE3804E.ADA
CE3804F.ADA	CE3804G.ADA	CE3804I.ADA	CE3804K.ADA
CE3804M.ADA	CE3805A.ADA	CE3805B.ADA	CE3806A.ADA
CE3806C.ADA	CE3806D.ADA	CE3806E.ADA	CE3809A.ADA
CE3809B.ADA	CE3810A.ADA	CE3901A.ADA	CE3905A.ADA
CE3905B.ADA	CE3905C.ADA	CE3905L.ADA	CE3906A.ADA
CE3906B.ADA	CE3906C.ADA	CE3906D.ADA	CE3906E.ADA
CE3906F.ADA	CE3908A.ADA	E28005C.ADA	

In addition, the following support files and tests for support files were modified for Version 1.10:

MACRO.DFS	REPSPEC.ADA	REPBODY.ADA	CZ1101A.ADA
CZ1102.ADA			

3 TEST DELETIONS

The following 26 tests appear in ACVC Version 1.9, but not in ACVC Version 1.10. Several of the deleted tests were renamed to new test names in ACVC 1.10 to conform to the naming conventions provided by the Implementer's Guide or to reflect a change in test class.

A37312A.ADA	A41307A.ADA	A41307C.ADA	B33006B.ADA
B34001S.ADA	B86001C.DEP	B86001D.TST	B97102A.ADA
BC10AGA.ADA	C35A03E.ADA	C35A03R.ADA	C900ACA.ADA
C910BDA.ADA	C910BDB.ADA	C950CHC.ADA	CE2210A.ADA
CE3203A.ADA	CE3301B.ADA	CE3301C.ADA	CE3402B.ADA
CE3405B.ADA	CE3409F.ADA	CE3410F.ADA	CE3412C.ADA
E24101A.TST	E66001D.ADA		

4 NEW TESTS

The following 623 tests have been added to Version 1.10 of the ACVC Test Suite. A test is considered "new" if the test name did not exist in ACVC Version 1.9. Several of the new tests were renamed from former tests in ACVC Version 1.9 to conform to the naming conventions provided by the Implementer's Guide or to reflect a change in test class.

A98001A.ADA	AD1009M.DEP	AD1009V.DEP	AD1009W.DEP
AD1C04D.DEP	AD3014C.DEP	AD3014F.DEP	AD3015C.DEP
AD3015F.DEP	AD3015H.DEP	AD3015K.DEP	AD7001B.ADA
AD7005A.ADA	AD7006A.ADA	AD7101A.ADA	AD7101C.ADA
AD7102A.ADA	AD7103A.ADA	AD7103C.ADA	AD7104A.ADA
B22003B.ADA	B24204D.ADA	B24204E.ADA	B24204F.ADA
B2A003D.ADA	B2A003E.ADA	B2A003F.ADA	B33002A.ADA
B33301B.ADA	B34005B.ADA	B34005E.ADA	B34005H.ADA
B34005K.ADA	B34005N.ADA	B34005Q.ADA	B34005T.ADA
B34006B.ADA	B34006E.ADA	B34006H.ADA	B34006K.ADA
B34014B.ADA	B34014D.ADA	B34014F.ADA	B34014I.ADA
B34014K.ADA	B34014M.ADA	B34014O.ADA	B34014Q.ADA
B34014S.ADA	B34014V.ADA	B34014X.ADA	B34014Z.ADA
B37201B.ADA	B37212B.ADA	B41307B.ADA	B44001B.ADA
B64001B.ADA	B64110A.ADA	B66001D.ADA	B67001H.ADA
B67001I.ADA	B67001J.ADA	B67001K.ADA	B85013D.ADA
B86001C.ADA	B86001D.ADA	B86001E.ADA	B86001F.ADA
B86001G.ADA	B86001H.ADA	B86001I.ADA	B86001J.ADA
B86001K.ADA	B86001L.ADA	B86001M.ADA	B86001N.ADA
B86001O.ADA	B86001P.ADA	B86001Q.ADA	B86001R.ADA
B86001S.ADA	B86001T.DEP	B86001U.DEP	B86001V.DEP
B86001W.DEP	B86001X.TST	B86001Y.TST	B86001Z.TST
B91003E.ADA	B95001C.ADA	B95069D.ADA	B95069E.ADA
B97101G.ADA	B97101H.ADA	BC3102C.ADA	BC3123C.ADA
BD5001A.ADA	BD5002A.ADA	BD5002B.ADA	BD5002C.ADA

BD5002D.ADA BD5002E.ADA BD5002F.ADA BD5005B.ADA
BD5006B.ADA BD5006C.ADA BD5006D.ADA BD5006J.ADA
BD5008A.ADA BD5101A.ADA BD5101B.ADA BD5102A.ADA
BD5103A.ADA BD5104A.ADA BE3301C.ADA C32111B.ADA
C32115B.ADA C34004C.ADA C34005A.ADA C34005C.ADA
C34005D.ADA C34005F.ADA C34005G.ADA C34005I.ADA
C34005J.ADA C34005L.ADA C34005M.ADA C34005O.ADA
C34006A.ADA C34006D.ADA C34006F.ADA C34007J.ADA
C34009A.ADA C34014A.ADA C34014C.ADA C34014E.ADA
C34014G.ADA C34014H.ADA C34014J.ADA C34014L.ADA
C34014N.ADA C34014P.ADA C34014R.ADA C34014T.ADA
C34014U.ADA C34014W.ADA C34014Y.ADA C35505E.ADA
C35505F.ADA C35A06O.ADA C35A06P.ADA C35A06Q.ADA
C35A06R.ADA C35A06S.ADA C37213K.ADA C37213L.ADA
C37312A.ADA C41307A.ADA C41307C.ADA C41309A.ADA
C91004B.ADA C91004C.ADA C95040B.ADA C98001B.ADA
C98001C.ADA CC1221B.ADA CC1221C.ADA CC1221D.ADA
CC1223A.ADA CC3123A.ADA CC3123B.ADA CC3125B.ADA
CC3125C.ADA CC3125D.ADA CC3126A.ADA CC3127A.ADA
CC3220A.ADA CC3221A.ADA CC3222A.ADA CC3223A.ADA
CC3606A.ADA CD1009A.DEP CD1009B.DEP CD1009C.DEP
CD1009D.DEP CD1009E.DEP CD1009F.DEP CD1009G.DEP
CD1009H.DEP CD1009I.DEP CD1009J.DEP CD1009K.DEP
CD1009L.DEP CD1009N.DEP CD1009O.DEP CD1009P.DEP
CD1009Q.DEP CD1009R.DEP CD1009S.DEP CD1009T.DEP
CD1009U.DEP CD1009X.DEP CD1009Y.DEP CD1009Z.DEP
CD1C03A.DEP CD1C03B.ADA CD1C03C.TST CD1C03E.DEP
CD1C03F.DEP CD1C03G.DEP CD1C03H.DEP CD1C03I.ADA
CD1C04A.DEP CD1C04B.DEP CD1C04C.DEP CD1C04E.DEP
CD1C06A.DEP CD1D01A.ADA CD1D01B.ADA CD1D01C.ADA
CD1D01D.ADA CD1D02A.ADA CD1D03A.ADA CD2A21A.DEP
CD2A21B.DEP CD2A21C.DEP CD2A21D.DEP CD2A21E.DEP
CD2A22A.DEP CD2A22B.DEP CD2A22C.DEP CD2A22D.DEP
CD2A22E.DEP CD2A22F.DEP CD2A22G.DEP CD2A22H.DEP
CD2A22I.DEP CD2A22J.DEP CD2A23A.DEP CD2A23B.DEP
CD2A23C.DEP CD2A23D.DEP CD2A23E.DEP CD2A24A.DEP
CD2A24B.DEP CD2A24C.DEP CD2A24D.DEP CD2A24E.DEP
CD2A24F.DEP CD2A24G.DEP CD2A24H.DEP CD2A24I.DEP
CD2A24J.DEP CD2A31A.DEP CD2A31B.DEP CD2A31C.DEP
CD2A31D.DEP CD2A32A.DEP CD2A32B.DEP CD2A32C.DEP
CD2A32D.DEP CD2A32E.DEP CD2A32F.DEP CD2A32G.DEP
CD2A32H.DEP CD2A32I.DEP CD2A32J.DEP CD2A41A.DEP
CD2A41B.DEP CD2A41C.DEP CD2A41D.DEP CD2A41E.DEP
CD2A42A.DEP CD2A42B.DEP CD2A42C.DEP CD2A42D.DEP
CD2A42E.DEP CD2A42F.DEP CD2A42G.DEP CD2A42H.DEP
CD2A42I.DEP CD2A42J.DEP CD2A51A.DEP CD2A51B.DEP
CD2A51C.DEP CD2A51D.DEP CD2A51E.DEP CD2A52A.DEP
CD2A52B.DEP CD2A52C.DEP CD2A52D.DEP CD2A52G.DEP
CD2A52H.DEP CD2A52I.DEP CD2A52J.DEP CD2A53A.DEP
CD2A53B.DEP CD2A53C.DEP CD2A53D.DEP CD2A53E.DEP
CD2A54A.DEP CD2A54B.DEP CD2A54C.DEP CD2A54D.DEP
CD2A54G.DEP CD2A54H.DEP CD2A54I.DEP CD2A54J.DEP

CD2A61A.DEP	CD2A61B.DEP	CD2A61C.DEP	CD2A61D.DEP
CD2A61E.DEP	CD2A61F.DEP	CD2A61G.DEP	CD2A61H.DEP
CD2A61I.DEP	CD2A61J.DEP	CD2A61K.DEP	CD2A61L.DEP
CD2A62A.DEP	CD2A62B.DEP	CD2A62C.DEP	CD2A62D.DEP
CD2A63A.DEP	CD2A63B.DEP	CD2A63C.DEP	CD2A63D.DEP
CD2A64A.DEP	CD2A64B.DEP	CD2A64C.DEP	CD2A64D.DEP
CD2A65A.DEP	CD2A65B.DEP	CD2A65C.DEP	CD2A65D.DEP
CD2A66A.DEP	CD2A66B.DEP	CD2A66C.DEP	CD2A66D.DEP
CD2A71A.DEP	CD2A71B.DEP	CD2A71C.DEP	CD2A71D.DEP
CD2A72A.DEP	CD2A72B.DEP	CD2A72C.DEP	CD2A72D.DEP
CD2A73A.DEP	CD2A73B.DEP	CD2A73C.DEP	CD2A73D.DEP
CD2A74A.DEP	CD2A74B.DEP	CD2A74C.DEP	CD2A74D.DEP
CD2A75A.DEP	CD2A75B.DEP	CD2A75C.DEP	CD2A75D.DEP
CD2A76A.DEP	CD2A76B.DEP	CD2A76C.DEP	CD2A76D.DEP
CD2A81A.TST	CD2A81B.TST	CD2A81C.TST	CD2A81D.TST
CD2A81E.TST	CD2A81F.TST	CD2A81G.TST	CD2A83A.TST
CD2A83B.TST	CD2A83C.TST	CD2A83E.TST	CD2A83F.TST
CD2A83G.TST	CD2A84B.DEP	CD2A84C.DEP	CD2A84D.DEP
CD2A84E.DEP	CD2A84F.DEP	CD2A84G.DEP	CD2A84H.DEP
CD2A84I.DEP	CD2A84K.DEP	CD2A84L.DEP	CD2A84M.DEP
CD2A84N.DEP	CD2A87A.TST	CD2A91A.TST	CD2A91B.TST
CD2A91C.TST	CD2A91D.TST	CD2A91E.TST	CD2A95A.TST
CD2B11B.DEP	CD2B11C.DEP	CD2B11D.DEP	CD2B11E.DEP
CD2B11F.DEP	CD2B11G.DEP	CD2B15B.DEP	CD2B15C.DEP
CD2B16A.DEP	CD2C11A.DEP	CD2C11B.DEP	CD2C11C.DEP
CD2C11D.DEP	CD2C11E.DEP	CD2D11A.DEP	CD2D11B.DEP
CD2D13A.DEP	CD3014A.DEP	CD3014B.DEP	CD3014D.DEP
CD3014E.DEP	CD3015A.DEP	CD3015B.DEP	CD3015D.DEP
CD3015E.DEP	CD3015G.DEP	CD3015I.DEP	CD3015J.DEP
CD3015L.DEP	CD3021A.DEP	CD4031A.DEP	CD4041A.DEP
CD4051A.DEP	CD4051B.DEP	CD4051C.DEP	CD4051D.DEP
CD5003B.DEP	CD5003C.DEP	CD5003D.DEP	CD5003E.DEP
CD5003F.DEP	CD5003G.DEP	CD5003H.DEP	CD5003I.DEP
CD5007B.DEP	CD5011A.DEP	CD5011B.DEP	CD5011C.DEP
CD5011D.DEP	CD5011E.DEP	CD5011F.DEP	CD5011G.DEP
CD5011H.DEP	CD5011I.DEP	CD5011K.DEP	CD5011L.DEP
CD5011M.DEP	CD5011N.DEP	CD5011O.DEP	CD5011Q.DEP
CD5011R.DEP	CD5011S.DEP	CD5012A.DEP	CD5012B.DEP
CD5012C.DEP	CD5012D.DEP	CD5012E.DEP	CD5012F.DEP
CD5012G.DEP	CD5012H.DEP	CD5012I.DEP	CD5012J.DEP
CD5012L.DEP	CD5012M.DEP	CD5013A.DEP	CD5013B.DEP
CD5013C.DEP	CD5013D.DEP	CD5013E.DEP	CD5013F.DEP
CD5013G.DEP	CD5013H.DEP	CD5013I.DEP	CD5013K.DEP
CD5013L.DEP	CD5013M.DEP	CD5013N.DEP	CD5013O.DEP
CD5013R.DEP	CD5013S.DEP	CD5014A.DEP	CD5014B.DEP
CD5014C.DEP	CD5014D.DEP	CD5014E.DEP	CD5014F.DEP
CD5014G.DEP	CD5014H.DEP	CD5014I.DEP	CD5014J.DEP
CD5014K.DEP	CD5014L.DEP	CD5014M.DEP	CD5014N.DEP
CD5014O.DEP	CD5014R.DEP	CD5014S.DEP	CD5014T.DEP
CD5014U.DEP	CD5014V.DEP	CD5014W.DEP	CD5014X.DEP
CD5014Y.DEP	CD5014Z.DEP	CD7003A.TST	CD7004A.TST
CD7004C.TST	CD7004D.TST	CD7005B.TST	CD7005E.TST

CD7006B.TST	CD7006E.TST	CD7007B.ADA	CD7007C.TST
CD7101B.TST	CD7101D.ADA	CD7101E.DEP	CD7101F.DEP
CD7101G.TST	CD7102B.TST	CD7103B.TST	CD7103D.TST
CD7104B.TST	CD7105A.ADA	CD7203B.ADA	CD7204B.ADA
CD7204C.DEP	CD7205C.DEP	CD7205D.DEP	CE2102L.ADA
CE2102M.ADA	CE2102N.ADA	CE2102O.ADA	CE2102P.ADA
CE2102Q.ADA	CE2102R.ADA	CE2102S.ADA	CE2102T.ADA
CE2102U.ADA	CE2102V.ADA	CE2102W.ADA	CE2102X.ADA
CE2102Y.ADA	CE2103C.ADA	CE2103D.ADA	CE2107L.ADA
CE2108E.ADA	CE2108F.ADA	CE2108G.ADA	CE2108H.ADA
CE2110D.ADA	CE2111F.ADA	CE2111I.ADA	CE2201H.ADA
CE2201I.ADA	CE2201J.ADA	CE2201K.ADA	CE2201L.ADA
CE2201M.ADA	CE2201N.ADA	CE2204C.ADA	CE2204D.ADA
CE2205A.ADA	CE2401I.ADA	CE2401J.ADA	CE2401K.ADA
CE2401L.ADA	CE2404B.ADA	CE2407B.ADA	CE2408B.ADA
CE2409B.ADA	CE2410B.ADA	CE3102D.ADA	CE3102E.ADA
CE3102F.ADA	CE3102G.ADA	CE3102H.ADA	CE3102I.ADA
CE3102J.ADA	CE3102K.ADA	CE3104B.ADA	CE3104C.ADA
CE3107B.ADA	CE3112C.ADA	CE3112D.ADA	CE3404D.ADA
CE3411B.ADA	CE3604B.ADA	CE3801B.ADA	CE3804H.ADA
CE3804J.ADA	CE3804L.ADA	CE3804N.ADA	CE3804O.ADA
CE3804P.ADA	CE3806B.ADA	CE3806F.ADA	CE3806G.ADA
CE3806H.ADA	CE3810B.ADA	E24201A.TST	ED1D04A.DEP
ED2A26A.DEP	ED2A56A.DEP	ED2A86A.TST	ED7004B.TST
ED7005C.TST	ED7005D.TST	ED7006C.TST	ED7006D.TST
EE3203A.ADA	EE3301B.ADA	EE3402B.ADA	EE3405B.ADA
EE3409F.ADA	EE3410F.ADA	EE3412C.ADA	

In addition, one new support unit (in one file), a system of programs (in 165 files) designed to aid in the validation process, and a macro substitution program have been added to ACVC Version 1.10:

SPPRT13.ADA	(The specification of a package whose body must be provided by the user.)
AVAT	(One package in files DATSPEC.ADA and DATBODY.ADA; all other units in files whose names have the form "AVAT%%%.ext" where each % is replaced by a letter or digit and "ext" is either "ADA" or "TST".)
MACROSUB	A program which modifies the "TST" tests inserting the values supplied in the file MACRO.DFS. MACRO.DFS must be modified to reflect the implementation dependent values of the compiler.

ACVC 1.10 Tape Information

1 INTRODUCTION

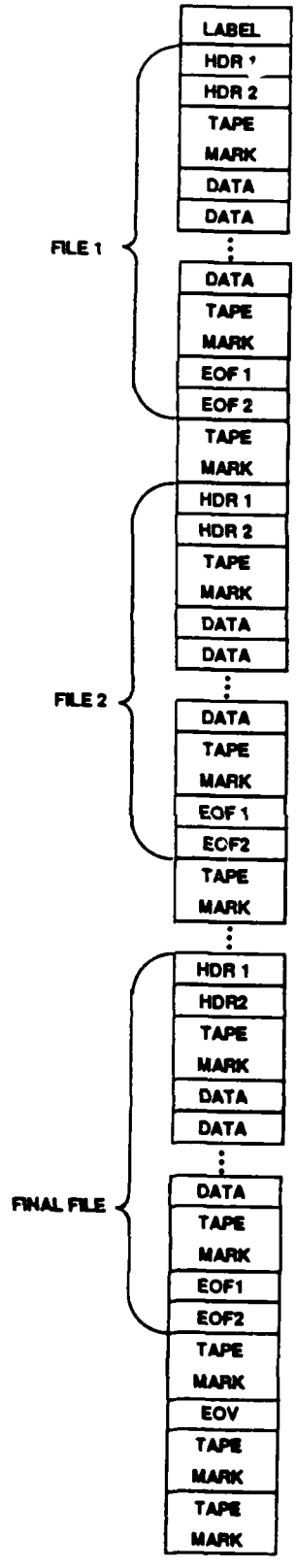
The ACVC tests are written in ANSI Standard format on one tape volume. There are 4358 test files in the ACVC Version 1.10. On the ANSI Standard format tape, there are a total of 266 files. Each of 39 of these files contains multiple tests packed onto a single file to save space on the tape volume. There are 59 single test files for tests containing non-graphical characters or long text lines; one Ada program to retrieve individual tests from the packed multi-test files; a system of programs (in 165 files) designed to aid in the validation process; a macro substitution program; and a list of all the ACVC Version 1.10 test files. The tape is written at a density of 1600 bpi.

2 TAPE FORMAT

The ACVC tests are written on one tape volume. Each file on the tape is written as a set of variable-length records in blocks of 2048 characters, using the ANSI variable-length record (D), blocked format. Each file in this format has the structure shown in Figure 1.

Each data block is 2048 characters in length. Each record within a block represents a line of text. The first four characters of each record make up the Record Control Word (RCW). The record length (the number of characters contained in the record) is expressed as a sequence of four ASCII characters occupying the entire RCW and yielding a decimal value. The record length includes the length of the RCW. Hence, a blank (empty) line is represented by the record 0004. A line containing the characters ADA would be represented as 0007ADA. Each record is completely contained within a block. If a record including the four-character RCW cannot be contained in the space remaining in a block, then each remaining byte in the block is set to a circumflex (^), and the record is placed in the next block. Figure 2 shows the format of a file that partially fills a data block.

ACVC DISTRIBUTION TAPE FORMAT



A-D8774

Figure 1.

```
0018-- C35702B.DEP00040050-- CHECK THAT LONG_FLOAT'DIGITS > FLOA
T'DIGITS00040020-- BAW 5 SEPT 800018-- SPS 2718/8300040016WITH R
EPORT;0024PROCEDURE C35702B IS00040020     USE REPORT;00040075BE
GIN TEST("C35702B","CHECK IF LONG_FLOAT HAS MORE DIGITS THAN FLO
AT");00040046      IF FLOAT'DIGITS >= LONG_FLOAT'DIGITS0070
 THEN FAILED("LONG_FLOAT HAS NOT MORE PRECISION THAN FLOAT");001
7      END IF;00040017      RESULT;00040016END C35702B;
```

Figure 2.

3 ACVC TEST FILES

The ACVC tests are stored on the tape using a method that packs many tests into one file. There are 39 multi-test-formatted files on the tape (named ACVC1...ACVC39), 35 separate files containing individual tests with special control characters, and 24 separate files containing individual tests with lines over 72 characters long. Files ACVC1...ACVC39 do not contain any text lines over 72 characters long. Also on the tape is an Ada program UNPACK that can be used on some systems to retrieve the individual ACVC tests from the packed files; a system of programs designed to aid in the validation process; a macro substitution program; and one file that contains the names of all the ACVC tests included in the files, ACVC10.LST. This is the order in which the files appear on the tape:

- File 1 : UNPACK.ADA
- File 2 : ACVC10.LST
- Files 3 - 41 : ACVC1...ACVC39
- Files 42 - 76 : Individual files with nongraphic ASCII characters
- Files 77 - 100 : Individual files with long lines
- File 101 : MACROSUB.ADA
- Files 102 - 266 : Files designed to aid in the validation process

Figure 3 shows how the files are structured on the tape. The multifile test files have this repeating structure:

- <<< test name
- an ACVC test
- >>> test name (blank) number of lines in the test

Figure 4 shows an example of the structure of the ACVC multifile format. The number of lines given at the end of each file following the >>> can be used to check that the file was unpacked properly.

UNPACK will work to separate the individual tests if the packed file read from the tape is in a format that conforms to the format used by the implementation of the Ada package TEXT_IO. UNPACK reads a packed file of tests and puts each of the tests into a single file. UNPACK requires as input the name of the file to be unpacked. New files are created using the ACVC test name as the file name. UNPACK will need to be run once for each packed file.

The program text may have to be modified to coincide with the file-naming conventions of the system onto which the tests will be loaded. The TEXT_IO.CREATE command on line 36 of the program UNPACK creates a file using the returned value from the function TESTFILE_NAME.

MULTIFILE FORMAT

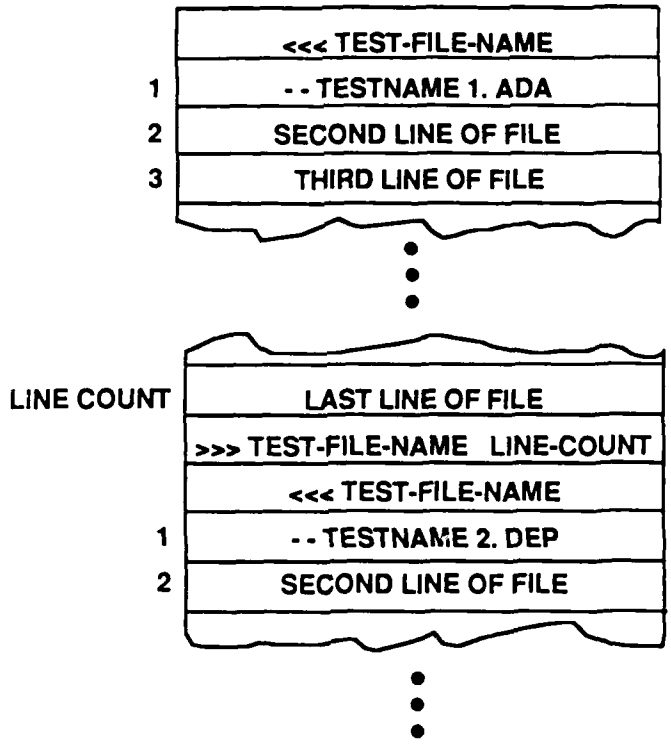


Figure 3.

```
<<< C35702B.DEP
- - C35702B.DEP

- - CHECK THAT LONG_FLOAT'DIGITS > FLOAT'DIGITS

- - BAW 5 SEPT 80
- - SPS 2/18/83

WITH REPORT;
PROCEDURE C35702B IS

    USE REPORT;

BEGIN TEST("C35702B","CHECK IF LONG_FLOAT HAS MORE DIGITS THAN FLOAT");

    IF FLOAT'DIGITS >= LONG_FLOAT'DIGITS
    THEN FAILED("LONG_FLOAT HAS NOT MORE PRECISION THAN FLOAT");
    END IF;

RESULT;

END C35702B;
>>> C35702B.DEP 21
```

Figure 4.

Modifications may be made inside this function to provide conformance to the file-naming conventions of the system. Without modification, program UNPACK will try to name the new files it creates with the names in the packed files that are listed after "<<< ". In figure 4, C35702B.DEP would be the name of the new file created which would not agree with the file naming conventions of some systems.

The specification, REPSPEC, and body, REPBODY, of package REPORT and the support unit SPPRT13SP.ADA are included in the packed file ACVC39. Package REPORT is used in many of the ACVC tests. Therefore, ACVC39 should be unpacked and REPSPEC, REPBODY, and SPPRT13SP compiled before any of the ACVC tests are run.

4 MULTIFILE TEST FILE CONTENTS

The following list shows each multifile test file with its corresponding number of individual files in it. This list may be helpful in unpacking the files. UNPACK will display the number of test files it retrieves from each packed file.

FILE	FILES CREATED
ACVC1	198
ACVC2	71
ACVC3	155
ACVC4	138
ACVC5	214
ACVC6	148
ACVC7	337
ACVC8	157
ACVC9	51
ACVC10	73
ACVC11	139
ACVC12	41
ACVC13	44
ACVC14	52
ACVC15	90
ACVC16	100
ACVC17	63
ACVC18	62
ACVC19	41
ACVC20	62
ACVC21	74
ACVC22	48
ACVC23	83
ACVC24	94
ACVC25	75
ACVC26	144
ACVC27	85
ACVC28	105
ACVC29	228
ACVC30	100
ACVC31	48
ACVC32	40
ACVC33	45
ACVC34	108
ACVC35	125
ACVC36	98
ACVC37	86
ACVC38	125
ACVC39	184

5 FILES WITH NONGRAPHIC CHARACTERS

Thirty-five of the files in the ACVC test suite contain ASCII control characters. These files are stored individually on the distribution tape. The following list shows the files with the corresponding control characters in the file.

A22006B.ADA	HT
A22006C.ADA	CR, VT, LF, FF
A22006D.ADA	HT
A22006E.ADA	CR, VT, LF, FF
A22006F.ADA	HT
B22005A.ADA	SOH
B22005B.ADA	STX
B22005C.ADA	ETX
B22005D.ADA	EOT
B22005E.ADA	ENQ
B22005F.ADA	ACK
B22005G.ADA	BEL
B22005H.ADA	BS
B22005I.ADA	NUL
B22005J.ADA	DEL
B22005K.ADA	FS, ESC
B22005L.ADA	GS, US
B22005M.ADA	RS
B22005N.ADA	SO
B22005O.ADA	SI
B22005P.ADA	DLE
B22005Q.ADA	DC1
B22005R.ADA	DC2
B22005S.ADA	DC3
B22005T.ADA	DC4
B22005U.ADA	NAK
B22005V.ADA	SYN
B22005W.ADA	ETB
B22005X.ADA	CAN
B22005Y.ADA	EM
B22005Z.ADA	SUB
B25002A.ADA	Control Characters SOH through BS, and SO through SUB, ESC, FS, GS, RS, US, NUL, DEL
B25002B.ADA	HT, VT, CR, LF, FF
B26005A.ADA	Control Characters SOH through SUB, ESC, FS, GS, RS, US, NUL, DEL
B27005A.ADA	Control Characters SOH through BS, and SO through SUB, NUL, ESC, FS, GS, RS, US, DEL

6 FILES WITH LONG TEXT LINES

Twenty-four of the files in the ACVC test suite contain lines longer than 72 characters. These files are stored individually on the distribution tape. The following table shows the files that have long lines along with the length of the longest line in the file.

TEST NAMES	LENGTH (characters)
C24113C.DEP	78
C24113D.DEP	86
C24113E.DEP	95
C24113F.DEP	106
C24113G.DEP	114
C24113H.DEP	123
C24113I.DEP	134
C24113J.DEP	142
C24113K.DEP	151
C24113L.DEP	162
C24113M.DEP	170
C24113N.DEP	179
C24113O.DEP	190
C24113P.DEP	198
C24113Q.DEP	207
C24113R.DEP	218
C24113S.DEP	226
C24113T.DEP	235
C24113U.DEP	247
C24113V.DEP	255
C24113W.DEP	264
C24113X.DEP	275
C24113Y.DEP	283
MACRO.DFS	87

7 ACVC TEST FILE NAMES

ACVC test file names may not conform to the naming conventions of the computer system on which the files are to be read. The test file names follow certain conventions. However, certain portions of a name provide information but may be removed to shorten a name while preserving uniqueness.

The maximum length of a test file descriptor is 13 characters, which consists of a file name and file type. The last four characters designate the file type--a period followed by three letters--and can be deleted without creating duplicate file names. When a file name consists of nine characters followed by a period, the ninth character is always M and can be deleted. All other test file names are seven or eight characters in length before a period occurs. Finally, the first letter of each test file name can be safely deleted without creating duplicate file names. If all these deletions are performed, test files will have names that are six to nine characters in length.

8 TEST MODIFICATION

The Ada tests will be periodically updated and corrected. Their correctness is not guaranteed either by the ACVC Maintenance Organization (AMO) or by its contractor. Please report errors in writing to:

Mr. Steve Wilson
ACVC Maintenance Organization
ASD/SCOL
Wright-Patterson AFB, OH 45433-6503

You will receive notification of changes to this version of the ACVC on a periodic basis. Please notify the AMO of any change of address.

1 Scope.

1.1 Identification.

This is the Ada Compiler Validation Capability (ACVC) User's Guide. This document details the use of Version 1.10 of the ACVC.

1.2 Purpose.

This user's guide describes the ACVC test suite, its use, and interpretation of the test results. It should be read by anyone who wishes to use the ACVC to test an implementation of the Ada language.

1.3 Introduction.

The primary purpose of the ACVC is to help decide whether an Ada translator/compiler conforms to ANSI/MIL-STD-1815A, the Reference Manual for the Ada Programming Language (RM). The RM is also known as the Ada Standard. The ACVC is maintained by the ACVC Maintenance Organization (AMO) under the direction of the Ada Validation Office (AVO) of the Air Force's Ada Joint Program Office (AJPO). The three main components of the ACVC are:

- An implementers' guide that describes the implementation implications of the RM and the conditions that are checked by each validation test program.
- A suite of validation test programs that are submitted to a compiler.
- Validation support tools that assist in determining system capabilities, preparing tests for compilation, and analyzing the results of execution.

2 References.

1. ANSI/MIL-STD-1815A (The Reference Manual for the Ada Programming Language), 22 January 1983.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., 1 December 1986.
4. ACVC Version 1.10 Version Description Document, SofTech, Inc., 1 December 1987.

5. ACVC Test Design and Coding Standards, SofTech, Inc., 1 December 1986.

3 Instruction For Use.

3.1 Ada Compiler Validation Capability Implementers' Guide.

All of the ACVC test programs are based on objectives given in the Ada Compiler Validation Capability Implementers' Guide (AIG). The AIG basically follows the RM. However, the AIG contains additional sections which further clarify the rules of the language, details the interpretation of any ambiguities in the RM, and presents objectives and guidelines for the tests.

3.2 Validation Test Programs.

The ACVC test suite is composed of various test programs designed to ensure that a compiler correctly conforms to all aspects of the RM. The test suite also attempts to ensure that the implementation does not add any illegal extentions to the language.

The goals of the ACVC include:

- Maximizing the amount of information derived from a validation attempt; in particular, identifying as many nonconformances as possible.
- Minimizing the effort needed to test a compiler; in particular, the effort needed to a) adapt the tests to a particular compiler environment, and b) determine which tests passed and which failed.
- Minimizing the effort needed to determine if a test is correct (that is, if it uses an appropriate algorithm to check the objective) when failed on a compiler.
- A thorough coverage of the language and the possible implementations of its features.
- Making the tests understandable and maintainable.

3.2.1 Test Naming Conventions.

Test names have the form <name>.<type>. The <name> component is composed of seven to nine characters. The interpretation of each character is described below, where the first column indicates the character position(s) and the second column gives the corresponding meaning:

1	Test classification (letter) - identifies the class of the test (see Classifications).
2	AIG chapter (hexadecimal) - identifies the chapter which contains the objective being tested.
3	Section (hexadecimal) - identifies the section within the above chapter.
4	Sub-section (alpha-numeric) - identifies the sub-section within the above section.
5, 6	Test objective (decimal) - identifies the objective, within the above sub-section, being tested.
7	Sub-objective (letter) - indicates a sub-objective of the above objective.
8	Compilation sequence (alpha-numeric) - indicates the compilation order of multiple files that comprise a single test (0 is compiled first). This position is present only if the test is composed of multiple files.
9	Main subprogram indication (character 'M') - indicates which file of a test set contains the main procedure. This position is present only for one file of a multi-file test.

The <type> component is a three-character extension which identifies the type of the test (see Type Extensions).

Characters 2-7 of the <name> component uniquely identify every test objective. Each individual objective has only a single classification and type designation associated with it. The classification and type of a test indicates its expected compilation/execution results.

3.2.1.1 Classifications.

The test classifications are A, B, C, D, E, and L. Tests that come under classes A, C, D, and E are also considered executable tests.

3.2.1.1.1 Class A.

Class A tests ensure acceptance (compilation) of certain language constructs which cannot be verified at run time. There are no testing conditions within them. These tests are expected to execute successfully - they simply report 'PASSED'. An implementation is considered to have failed any test for which it does not indicate otherwise.

3.2.1.1.2 Class B.

Class B tests check that illegal constructs are caught and that there are no illegal extentions to the language. These tests are expected to fail compilation. They are graded as 'PASSED' if all of the indicated errors in the test are caught and no legal constructs are rejected. An implementation is considered to have failed any test for which an error is not caught or a legal construct is rejected.

Lines containing errors which must be caught are marked '-- ERROR:'. This error comment may also include a very brief indication of what makes the construct illegal. Lines containing legal constructs which must be accepted are marked '-- OK.'.

The following rules apply in satisfying the pass criteria:

- The compiler rejects a test which contains only a single error.
- The compiler displays an error message for all of the '-- ERROR:' indications in a test with multiple errors.
- All lines marked '-- OK.' are accepted.
- Split tests can be created (see Modifying Tests).

3.2.1.1.3 Class C.

Class C tests check the run-time system and ensure that verifiable constructs are implemented correctly. These tests are expected to execute. The tests will indicate either 'PASSED', 'FAILED', or 'NOT_APPLICABLE' based on the conditions tested. An implementation is considered to have failed any test for which it does not indicate otherwise.

3.2.1.1.4 Class D.

Class D tests are capacity tests. These tests are expected to execute successfully if no errors are reported during compilation. If an error is reported during compilation, it must be an error associated with the capacity being tested, in which case the test is considered inapplicable. Otherwise, an implementation may be considered to have failed any test for which it does not indicate differently; such failure should be reported to the AVO for a ruling.

3.2.1.1.5 Class E.

Class E tests check implementation-dependent options and the resolution of ambiguities in the RM. These tests are expected to execute successfully. Tests which execute print comments to indicate how the options or ambiguities are handled. The source code for each test lists specific pass/fail criteria. For some tests, these criteria are indicated

at execution time. These criteria are compared to the comments to make a grading determination. An implementation is considered to have failed any test for which it does not indicate otherwise.

3.2.1.1.6 Class L.

Class L tests ensure that all library unit dependencies within a program are satisfied before said program can be linked and executed, and that circularity among units is not allowed. These tests are expected to fail at link time; however, failure to compile may be acceptable for some implementations. These tests are graded as 'PASSED' if they do not successfully complete the link phase. Any test which begins execution is graded as 'FAILED'.

For tests which might possibly fail at compile time, the lines containing the illegal constructs are marked '-- ERROR:'.

3.2.1.2 Type Extensions.

The type extensions and their meanings are:

ADA - These tests check the standard features of the language that must be implemented by all compilers. They are expected to be compiled/executed unmodified and must be successfully processed by all implementations.

DEP - These tests involve optional, implementation-dependent features of the language. Each test concerned with supported features must be successfully processed as indicated by its classification. Tests concerned with unsupported features are considered inapplicable, provided they are processed as described below:

- Class B tests should be successfully compiled.
- Executable tests must be rejected at compile time as described in the test's applicability criteria.

TST - These tests also check implementation-dependent features of the language. They contain macros for the substitution of certain implementation-dependent values (see Macro Substitutions). Generally, each of these tests is expected to be successfully processed as indicated by its classification. However, some of the tests may be concerned with unsupported features and therefore are treated as described for DEP tests.

3.2.2 Coding Standards.

To help meet the goals of the ACVC, certain coding standards are followed.

3.2.2.1 General.

Some general test writing standards include:

- Tests are written using the basic 55-character character-set (26 capital letters, 10 digits, and 19 punctuation marks).
- Maximum line length is 72 characters, except as noted in certain DEP and TST tests.
- Numeric values are generally in the range (-128..127). However, if necessary, they can be within the range (-2048..2047), a maximum of 12 bits.
- Tests are written using as few Ada features as are necessary to write a self-checking executable test that is not too difficult to read or modify.

To help ensure a thorough coverage of the language, and to promote understandability and maintainability, many objectives are divided into sub-objectives, with some checked in a single test file and others checked in separate test files. For example, if a test objective is concerned with some property of scalar types, then individual subtests check:

- INTEGER, CHARACTER, BOOLEAN, and a user-defined enumeration type, all in one test file since we expect all of these types to be implemented.
- FLOAT and a user-defined floating-point type in a separate test file.
- A fixed-point type in another separate test file.

Sub-objectives which are difficult to test are also tested in separate tests files.

3.2.2.2 Header Format.

The following header is placed at the beginning of all new and modified tests:

```
-- filename.

-- OBJECTIVE:
        .
        .
        .
-- SEPARATE FILES:
        .
        .
        .
-- APPLICABILITY CRITERIA:
        .
```

```
        .
        .
-- PASS/FAIL CRITERIA:
        .
        .
        .
-- MACRO SUBSTITUTIONS:
        .
        .
        .
-- HISTORY:
--      ccc  mm/dd/yy  comment about change.
        .
        .
        .
```

The purpose of each section is:

filename	- identical to the actual name of the file.
objective	- describes the test's objective and explains the exact conditions or sub-objectives presently being tested.
separate files	- identifies all of the component files of a multiple file test, with a brief description of each file.
applicability criteria	- details under what conditions the test can be ruled inapplicable.
pass/fail criteria	- explains how to interpret the run-time results for grading the test.
macro substitution	- identifies the macros in the test which must be substituted and a brief description of what the replacement value represents.
history	- details who has made changes to the test, when, and what those changes were.

The file name, objective and history sections are always present. The separate files, applicability criteria, pass/fail criteria and macro substitutions sections are present only for tests which have such conditions.

3.2.2.3 Test Body Format.

Note: test body refers to the code which is compiled/executed and is not necessarily synonymous with an Ada unit body.

The names of all library units correspond to the name of the test to which they are associated.

Class B tests are generally library procedures. Other units are used when required by the objective, in which case there is not necessarily a main subprogram.

The main subprogram of all executable tests is a procedure. The name of this procedure is identical to the name of the test.

3.2.2.3.1 Test Initialization/Reporting And Test Objective Blocks.

The test body of an executable test has the following logical execution sequence:

1. Call to REPORT.TEST, which prints the test identifier and description.

2. Elaboration of objective-related declarations.

3. Testing of the objective - calls to REPORT.FAILED and REPORT.NOT_APPLICABLE as appropriate.

4. Call to REPORT.RESULT, which prints the pass/fail message.

The tests generally consist of a single procedure with the objective being tested within an inner DECLARE block. If the test objective is divided into multiple sub-objectives, each sub-objective is tested within a separate DECLARE block. For objectives which require packages as library units, package declarations and elaborations are arranged to result in the previously described execution sequence.

3.2.2.3.2 Ada Unit Bodies.

The format of the statements and language constructs is designed to make the syntactic and semantic structure of the test program apparent. In an attempt to defeat high-quality optimizing compilers, which may execute or remove the critical portion of a test at compile time, certain coding practices have been devised. These practices include calling identity functions, using objects in statements which appear to have no significance, and using IF-THEN-ELSE statements instead of just a simple IF statement.

3.2.3 Macro Substitutions.

TST type tests contain macros which represent implementation-dependent characteristics of the language. The macros are given as identifiers which begin with a dollar sign ($). Each macro must be replaced with an appropriate textual value to make the tests suitable for compilation.

The substitutions can be done automatically by using the MACROSUBS program distributed with the ACVC. This program will request the replacement value for each macro and then edit all the TST tests in the suite to make the substitution.

Appendix A contains a description of all of the macros used in the test suite.

3.2.4 Processing The ACVC Test Programs.

Processing of the test suite consists of:

1. Compiling class B tests.

2. Compiling and linking class L tests.

3. Compiling, linking, and executing class A, C, D, and E tests.

It is recommended that the suite be processed by chapter. Generally, the test suite is processed within a single program library which must be initialized as described below.

3.2.4.1 Prerequisites.

The following prerequisites must be met before processing the test suite:

1. All of Ada must be implemented - the test suite attempts to cover the entire Ada language.

2. The support package REPORT must be compiled, checked out, and loaded into the program library (see Package Report).

3. The support procedure CHECKFILE must be compiled, checked out, and loaded into the program library (see Procedure Checkfile).

4. The support package SPPRT13 must be compiled and loaded into the program library (see Package Spprt13).

3.2.4.2 Dependencies Among Tests.

Test files which use character position 8 (compilation sequence) in their names have an inherent dependency. These test files must be compiled successively and, for class A, C, D, E, and L tests, linked together.

Some of the executable file I/O tests create a data file that is to be used by a subsequent test. In these instances, the tests must be run in sequence, without deleting the data file that was created. The tests which create such a file, and the tests that depend on that file are:

CE2108A must be followed by CE2108B.
CE2108C must be followed by CE2108D.
CE3112A must be followed by CE3112B.

3.2.4.3 Test Applicability.

Certain tests in the suite are concerned with optional, implementation-dependent features of the language. Hence, some of these tests may not be applicable to all implementations. The applicability of each test is based on the criteria given in the test file. The handling of tests concerned with unsupported features is described above under Type Extensions.

For executable tests, lines containing these implementation-dependent features are marked "-- NA => ERROR.". If an implementation does not support the specified feature, then these constructs must be rejected. If the constructs so indicated are rejected, then the test is considered inapplicable - any other errors reported are ignored. If the constructs so indicated are not rejected, then the test remains applicable and must execute successfully.

All the implementation-dependent tests are compiled, and executed if compiled successfully, during a validation attempt - except those tests which require a floating-point DIGITS value that exceeds SYSTEM.MAX_DIGITS. All results for a particular feature must be consistent (i.e., all 'PASSED' or all 'NOT-APPLICABLE').

Appendix B contains the test applicability criteria for all the implementation-dependent tests.

3.2.4.4 Test Disputes.

The Certification Body has defined "dispute" to mean:

> For the purposes of validation, a dispute is constituted by any behavior of the candidate Ada implementation that is not explicitly permitted by the code of the ACVC tests, by the documentation from the AMO that accompanies the ACVC tapes, or by other documentation provided to the vendor by the Ada Validation Facility (AVF).

Disputes must be brought to the attention of the AVF as soon as possible so that they can be resolved by the AVF and AVO. A dispute related to a validation attempt should be sent to the AVF handling the validation. Other disputes of ACVC tests can be sent to the AMO.

3.2.4.5 Guidelines For Modifying Tests.

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made with the approval of the AVO, and are made in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include:

- Adding a length clause to alter the default size of a collection.
- Creating split tests.
- Confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

No original test can be modified; copies must be made. It is recommended that the copies be named by appending an alpha-numeric character to the <name> portion of the original test file. If possible, the line numbers of the original test should remain intact.

Split tests can be created for class B and L tests. Split tests have certain lines commented out to facilitate catching any previously undetected errors. Split tests can also be used to ensure that all legal constructs are passed.

3.3 Validation Support Tools.

The following validation tools are used to a) provide additional information about an implementation, and b) reduce the effort required to perform a validation attempt.

3.3.1 The Ada Validation Assistant Tool.

3.3.1.1 Description.

The Ada Validation Assistant Tool (AVAT) is a set of five programs that test implementation-dependent features and produce lists of possibly inapplicable tests. Although it is not part of the test suite, the AVAT system can assist in the validation process by providing information about the implementation.

AVAT attempts to determine whether certain features of the language are supported by the implementation. These features include predefined types other than those required by the language and certain forms of representation clauses. AVAT also attempts to measure the degree to which some features are supported, including fixed- and floating-point precision, and compilation of generic specifications and bodies in separate files. Finally, AVAT reports the values of many implementation-dependent values, including integer type bounds, real type accuracies and ranges, and the constants declared in the package SYSTEM.

For the features of the language that are not (fully) supported, the AVAT programs report lists of ACVC tests that might be inapplicable due to the lack of (full) support. These lists may not be complete; that is, there may be unlisted tests which are inapplicable due to an implementation's lack of support for a particular feature. Likewise, the lists may not be entirely accurate for a particular implementation; that is, due to implementation decisions not anticipated by AVAT's designers, a test may be listed when it is actually applicable to the implementation. The final determination of the applicability of a test depends on the specific criteria given in the text of the test, the agreement of test

results with the vendor's documentation, and, in some cases, the vendor's explanation of unusual test results. It is the vendor's responsibility to reconcile any inconsistencies; for validation purposes, justification of inconsistencies must be provided to the AVF in the form of a test dispute.

Each AVAT program consists of a data collection and reporting package (common to the five AVAT programs), a number of auxiliary library procedures, and a main procedure which imports the package and invokes the auxiliary procedures. Some of the auxiliary procedures simply provide section headers for the report, and others provide the values of system-defined constants. The remaining procedures (the majority) are used to determine whether specific language features are supported. There are two or more versions of each such procedure. For example, the procedure which tests for support of 'SIZE clauses for integer types occurs in four versions. The first version contains no 'SIZE clauses; if it is executed, it stores (in a structure declared in the package) a message that 'SIZE clauses are not supported for integer types, and a list of ACVC tests that are therefore potentially inapplicable. The second version contains a 'SIZE clause specifying the implementation's default size; if it executes, it stores a message that only default size specifications are accepted, and a corresponding list of potentially inapplicable ACVC tests. Likewise, the third and fourth versions contain intermediate and minimal size specifications and store corresponding messages and lists of tests. The four versions of the procedure must be submitted to the compiler in the given order. The last one to successfully compile will be the version invoked when the main program executes; thus the appropriate message and test list will be reported.

The AVAT programs are organized as follows:

COMMON PACKAGE DATACOLLECTION

This package collects and reports the messages and lists of potentially inapplicable tests.

Specification	DATSPEC.ADA
Body	DATBODY.ADA

PART ONE AVATNUM

This program reports on numeric type declarations and characteristics.

Main procedure	AVATNUM.ADA
Auxiliary procedures	AVAT000.ADA .. AVAT006.ADA
	AVAT007.TST
	AVAT008.ADA .. AVAT012.ADA
	AVAT013.TST
	AVAT014.ADA
	AVAT015.TST
	AVAT016.ADA .. AVAT026.ADA

PART TWO AVATLEN

This program reports on the support for length clauses.

Main procedure	AVATLEN.ADA
Auxiliary procedures	AVAT100.ADA .. AVAT134.ADA
	AVAT135.TST
	AVAT136.ADA
	AVAT137.TST
	AVAT138.ADA .. AVAT154.ADA

PART THREE AVATREP

This program reports on the support for enumeration and record representation clauses and address clauses. Some of the auxiliary procedures in this program depend on the support package SPPRT13.

Main procedure	AVATREP.ADA
Auxiliary procedures	AVAT200.ADA .. AVAT227.ADA

PART FOUR AVATGEN

This program reports on the support for separate compilation of generic specifications, bodies, and subunits, and for file I/O.

Main procedure	AVATGEN.ADA
Auxiliary procedures	AVAT300.ADA .. AVAT335.ADA

PART FIVE AVATSTD

This program reports the values of constants and type characteristics in the packages STANDARD and SYSTEM.

Main procedure	AVATSTD.ADA
Auxiliary procedures	AVAT400.ADA .. AVAT411.ADA

3.3.1.2 Compilation.

The five programs that make up AVAT may be processed in any order; however, the order of compilation within each program is crucial. The specification of package DATACOLLECTION must be compiled as the first unit in each program; the body, of course, may be compiled at any subsequent time before the link step. The auxiliary procedures must be compiled after the package and before the main procedure, since they import the package and the main procedure imports the auxiliary procedures. It is especially important to note that the auxiliary procedures must be compiled in the order given by their file names, since the success of the programs depends on the required library management. Note that this compilation order requirement prohibits the use of any sort of concurrent or interleaved compilation, including the use of batch queues with more than one job running at the same time.

3.3.1.3 Interpretation Of Results.

When the five AVAT programs execute, they report the values of implementation-dependent constants, report that certain features are (are not) supported, and produce lists of tests which are potentially inapplicable due to the apparent lack of support of certain features. The reports should be read and carefully considered in light of the system documentation and known implementation decisions. The lists of potentially inapplicable tests serve as a guide in determining applicability, but final decisions are made on the basis of test behavior, test-specific applicability criteria, and consistency with documentation.

3.3.2 Library Support Units.

The following library support units must be compiled and loaded into the program library that is used when compiling and linking the ACVC tests.

3.3.2.1 Reporting Test Results.

3.3.2.1.1 Package Report.

All executable tests use the REPORT support package. This package contains routines to 1) automate reporting of test results and 2) help prevent optimizing out any test code. Automation of reporting test results is accomplished with a set of procedures which identify the test and indicate its result. Optimization is defeated with a set of identity functions.

Package REPORT uses most of the Ada features used throughout the test suite, namely:

```
type INTEGER,   literals, +, -, relations, :=
type CHARACTER, literals, &, relations, :=
type STRING,    literals, &, relations, :=
type BOOLEAN,   literals, AND, OR, NOT, relations, :=

user-defined enumeration type,  literals, relations, :=

object declarations (and initialization) for the above types.

subprogram declarations, bodies, and calls, with formal
parameters of all modes and of the above types.

IF statements.


Executable tests generally follow the format:

WITH REPORT; USE REPORT;
PROCEDURE Testname IS
     ...
BEGIN
```

```
        TEST ("Testname", "Description ...");
             ...
        IF ... /= CorrectValue THEN
             FAILED ("Reason");
        END IF;
             ...
        RESULT;
    END Testname;
```

The specification of package REPORT is given in the file REPSPEC.ADA. The body of package REPORT is given in the file REPBODY.ADA. The specification and body of package REPORT are also given in Appendix C.

The acceptance tests CZ1101A.ADA and CZ1102A.ADA must be compiled and executed before a validation attempt to ensure correct operation of REPORT. A listing of the output for CZ1101A.ADA is given in Appendix E.

3.3.2.1.2 Allowable Modifications To Package Report.

Since the body of package REPORT is independent of the tests themselves, it can be modified to use only those implementation features that are actually working. Modifications made for a validation attempt must be presented to the AVF.

3.3.2.2 File I/O.

3.3.2.2.1 Procedure CheckFile.

Some of the executable file-I/O tests (CE... tests) use a checking procedure called CHECKFILE. CHECKFILE is used to determine the implementation's text file characteristics.

The source code for this procedure is found in the file CHECKFILE.ADA. The body of procedure CHECKFILE is also given in Appendix D.

The acceptance test CZ1103A.ADA must be compiled and executed before a validation attempt to ensure correct operation of CHECKFILE. CZ1103A checks that errors in text files are properly detected; therefore, CZ1103A will print some failure messages when it is executed. The presence of these messages does not mean the test has failed. A listing of the output for CZ1103A.ADA is given in Appendix E.

3.3.2.3 Address Clauses.

3.3.2.3.1 Package Spprt13.

Package SPPRT13 contains functions which return values of type SYSTEM.ADDRESS for use in ADDRESS clauses. Only the specification of this package is distributed with the ACVC; the body must be supplied by the implementer. Spprt13 is required for running test B91001H.ADA, many chapter 13 tests (CD... tests), and AVAT.

3.3.2.3.2 Allowable Modifications To Package Spprt13.

If the type SYSTEM.ADDRESS is defined to be static, then the functions in package SPPRT13 can be replaced with constants. Modifications made for a validation attempt must be presented to the AVF.

APPENDIX A

Macros and Their Descriptions

Following is the information contained in the MACRO.DFS file. Each macro definition includes:

1. The macro itself.
2. A description of what the replacement value represents.
3. A list of tests which depend on the macro.
4. An example replacement value.

```
-- $MAX_IN_LEN
-- AN INTEGER LITERAL GIVING THE MAXIMUM LENGTH PERMITTED BY THE
-- COMPILER FOR A LINE OF ADA SOURCE CODE (NOT INCLUDING AN END-OF-LINE
-- CHARACTER).
-- USED IN:  A26007A
MAX_IN_LEN           60

-- $BIG_ID1
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- USED IN:  C23003A  C23003B  C23003C  B23003D  B23003E  C23003G
--           C23003H  C23003I  C23003J  C35502D  C35502F
BIG_ID1
             AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA1

-- $BIG_ID2
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID1 ONLY IN THE LAST CHARACTER.
-- USED IN:  C23003A  C23003B  C23003C  B23003F  C23003G  C23003H
--           C23003I  C23003J
BIG_ID2
             AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA2
```

```
-- $BIG_ID3
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- USED IN:  C23003A  C23003B  C23003C  C23003G  C23003H  C23003I
--           C23003J
BIG_ID3
              AAAAAAAAA_AAAAAAAAA_AAAAAAAAA3AAAAAAAAA_AAAAAAAAA_AAAAAAAAAA

-- $BIG_ID4
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID3 ONLY IN THE MIDDLE CHARACTER.
-- USED IN:  C23003A  C23003B  C23003C  C23003G  C23003H  C23003I
--           C23003J
BIG_ID4
              AAAAAAAAA_AAAAAAAAA_AAAAAAAAA4AAAAAAAAA_AAAAAAAAA_AAAAAAAAAA

-- $BIG_STRING1
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING2
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN:  C35502D  C35502F
BIG_STRING1           "AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_"

-- $BIG_STRING2
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING1
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN:  C35502D  C35502F
BIG_STRING2           "AAAAAAAAA_AAAAAAAAA_AAAAAAAAA1"

-- $MAX_STRING_LITERAL
-- A STRING LITERAL CONSISTING OF $MAX_IN_LEN CHARACTERS (INCLUDING THE
-- QUOTE CHARACTERS).
-- USED IN:  A26007A
MAX_STRING_LITERAL
   "CCCCCCC10CCCCCCCCC20CCCCCCCCC30CCCCCCCCC40CCCCCCCCC50CCCCCCCCC59"

-- $NEG_BASED_INT
-- A BASED INTEGER LITERAL (PREFERABLY BASE 8 OR 16) WHOSE HIGHEST ORDER
-- NON-ZERO BIT WOULD FALL IN THE SIGN BIT POSITION OF THE
-- REPRESENTATION FOR SYSTEM.MAX_INT, I.E., AN ATTEMPT TO WRITE A
-- NEGATIVE VALUED LITERAL SUCH AS -2 BY TAKING ADVANTAGE OF THE
-- BIT REPRESENTATION.
-- USED IN:  E24201A
NEG_BASED_INT         16#FFFFFFFE#

$BIG_INT_LIT
-- AN INTEGER LITERAL WHOSE VALUE IS 298, BUT WHICH HAS
-- ($MAX_IN_LEN - 3) LEADING ZEROES.
-- USED IN:  C24003A
BIG_INT_LIT
         000000000000000000000000000000000000000000000000000000000298
```

```
$BIG_REAL_LIT
-- A UNIVERSAL_REAL LITERAL WHOSE VALUE IS 690.0, BUT WHICH HAS
-- ($MAX_IN_LEN - 5) LEADING ZEROES.
-- USED IN:  C24003B  C24003C
BIG_REAL_LIT
        000000000000000000000000000000000000000000000000000690.0

-- $MAX_LEN_INT_BASED_LITERAL
-- A BASED INTEGER LITERAL (USING COLONS) WHOSE VALUE IS 2:11:, HAVING
-- ($MAX_IN_LEN - 5) ZEROES BETWEEN THE FIRST COLON AND THE FIRST 1.
-- USED IN:  C2A009A
MAX_LEN_INT_BASED_LITERAL
    2:0000000000000000000000000000000000000000000000000011:

-- $MAX_LEN_REAL_BASED_LITERAL
-- A BASED REAL LITERAL (USING COLONS) WHOSE VALUE IS 16:F.E:, HAVING
-- ($MAX_IN_LEN - 7) ZEROES BETWEEN THE FIRST COLON AND THE F.
-- USED IN:  C2A009A
MAX_LEN_REAL_BASED_LITERAL
   16:0000000000000000000000000000000000000000000000000F.E:

-- $BLANKS
-- A SEQUENCE OF ($MAX_IN_LEN - 20) BLANKS.
-- USED IN:  B22001A  B22001B  B22001C  B22001D  B22001E  B22001F
--           B22001G  B22001I  B22001J  B22001K  B22001L  B22001M
--           B22001N
--      <       LIMITS OF SAMPLE SHOWN BY ANGLE BRACKETS          >
BLANKS

-- $MAX_DIGITS
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_DIGITS.
-- USED IN:  B35701A  CD7102B
MAX_DIGITS          6

-- $NAME
-- THE NAME OF A PREDEFINED INTEGER TYPE OTHER THAN INTEGER,
-- SHORT_INTEGER, OR LONG_INTEGER.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS   NO_SUCH_TYPE_AVAILABLE.)
-- USED IN:  AVAT007  C45231D  B86001X
NAME  TINY_INTEGER

-- $FLOAT_NAME
-- THE NAME OF A PREDEFINED FLOATING POINT TYPE OTHER THAN FLOAT,
-- SHORT_FLOAT, OR LONG_FLOAT.  (IMPLEMENTATIONS WHICH HAVE NO SUCH
-- TYPES SHOULD USE AN UNDEFINED IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT013  B86001Y
FLOAT_NAME          SHORT_SHORT_FLOAT
```

```
-- $FIXED_NAME
-- THE NAME OF A PREDEFINED FIXED POINT TYPE OTHER THAN DURATION.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT030  B86001Z
FIXED_NAME            NO_SUCH_FIXED_TYPE

-- $INTEGER_FIRST
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS INTEGER'FIRST.
-- USED IN:  C35503F  B54B01B
INTEGER_FIRST         -32_768

-- $INTEGER_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST.
-- USED IN:  C35503F  B45232A  B45B01B
INTEGER_LAST          32_767

-- $INTEGER_LAST_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST + 1.
-- USED IN:  C45232A
INTEGER_LAST_PLUS_1   32_768

-- $MIN_INT
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS SYSTEM.MIN_INT.
-- THE LITERAL MUST NOT CONTAIN UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503D  C35503F  CD7101B
MIN_INT               -2147483648

-- $MAX_INT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT.
-- THE LITERAL MUST NOT INCLUDE UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503D  C35503F  C4A007A  CD7101B
MAX_INT               2147483647

-- $MAX_INT_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT + 1.
-- USED IN:  C45232A
MAX_INT_PLUS_1        2_147_483_648

-- $LESS_THAN_DURATION
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE (NOT SUBJECT TO
-- ROUND-OFF ERROR IF POSSIBLE) LIES BETWEEN DURATION'BASE'FIRST AND
-- DURATION'FIRST.  IF NO SUCH VALUES EXIST, USE A VALUE IN
-- DURATION'RANGE.
-- USED IN:  C96005B
LESS_THAN_DURATION    -75_000.0
```

```
-- $GREATER_THAN_DURATION
-- A REAL LITERAL WHOSE VALUE (NOT SUBJECT TO ROUND-OFF ERROR
-- IF POSSIBLE) LIES BETWEEN DURATION'BASE'LAST AND DURATION'LAST.  IF
-- NO SUCH VALUES EXIST, USE A VALUE IN DURATION'RANGE.
-- USED IN:  C96005B
GREATER_THAN_DURATION          75_000.0

-- $LESS_THAN_DURATION_BASE_FIRST
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE IS LESS THAN
-- DURATION'BASE'FIRST.
-- USED IN:  C96005C
LESS_THAN_DURATION_BASE_FIRST          -131_073.0

-- $GREATER_THAN_DURATION_BASE_LAST
-- A REAL LITERAL WHOSE VALUE IS GREATER THAN DURATION'BASE'LAST.
-- USED IN: C96005C
GREATER_THAN_DURATION_BASE_LAST        131_073.0

-- $COUNT_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.COUNT'LAST.
-- USED IN:  CE3002B
COUNT_LAST             32_767

-- $FIELD_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.FIELD'LAST.
-- USED IN:  CE3002C
FIELD_LAST             32_767

-- $ILLEGAL_EXTERNAL_FILE_NAME1
-- AN ILLEGAL EXTERNAL FILE NAME (E.G., TOO LONG, CONTAINING INVALID
-- CHARACTERS, CONTAINING WILD-CARD CHARACTERS, OR SPECIFYING A
-- NONEXISTENT DIRECTORY).
-- USED IN:  CE2103A  CE2102C  CE2102H  CE2103B  CE3102B  CE3107A
ILLEGAL_EXTERNAL_FILE_NAME1  \NODIRECTORY\FILENAME

-- $ILLEGAL_EXTERNAL_FILE_NAME2
-- AN ILLEGAL EXTERNAL FILE NAME, DIFFERENT FROM $EXTERNAL_FILE_NAME1.
-- USED IN:  CE2102C  CE2102H  CE2103A  CE2103B
ILLEGAL_EXTERNAL_FILE_NAME2  THIS-FILE-NAME-IS-TOO-LONG-FOR-MY-SYSTEM

-- $ACC_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE MINIMUM NUMBER OF BITS
-- SUFFICIENT TO HOLD ANY VALUE OF AN ACCESS TYPE.
-- USED IN:  CD1C03C  CD2A81A  CD2A81B  CD2A81C  CD2A81D  CD2A81E
--           CD2A81F  CD2A81G  CD2A83A  CD2A83B  CD2A83C  CD2A83E
--           CD2A83F  CD2A83G  ED2A86A  CD2A87A
ACC_SIZE               32
```

```
-- $TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS A SINGLE ENTRY WITH ONE INOUT PARAMETER.
-- USED IN:  CD2A91A  CD2A91B  CD2A91C  CD2A91D  CD2A91E
TASK_SIZE            128

-- $MIN_TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS NO ENTRIES, NO DECLARATIONS, AND "NULL;"
-- AS THE ONLY STATEMENT IN ITS BODY.
-- USED IN:  CD2A95A
MIN_TASK_SIZE        64

-- $NAME_LIST
-- A LIST OF THE ENUMERATION LITERALS IN THE TYPE SYSTEM.NAME, SEPARATED
-- BY COMMAS.
-- USED IN:  CD7003A
NAME_LIST            OUR_VMS_ADA, OUR_ULTRIX_ADA, OUR_ELN_ADA

-- $DEFAULT_SYS_NAME
-- THE VALUE OF THE CONSTANT SYSTEM.SYSTEM_NAME.
-- USED IN:  CD7004A  CD7004C  CD7004D
DEFAULT_SYS_NAME     OUR_VMS_ADA

-- $NEW_SYS_NAME
-- A VALUE OF THE TYPE SYSTEM.NAME, OTHER THAN $DEFAULT_SYS_NAME.  IF
-- THERE IS ONLY ONE VALUE OF THE TYPE, THEN USE THAT VALUE.
-- NOTE: IF THERE ARE MORE THAN TWO VALUES OF THE TYPE, THEN THE
--       PERTINENT TESTS ARE TO BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN:  ED7004B1
NEW_SYS_NAME         OUR_ULTRIX_ADA

-- $DEFAULT_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.STORAGE_UNIT.
-- USED IN:  CD7005B  ED7005D3M  CD7005E
DEFAULT_STOR_UNIT    16

-- $NEW_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA STORAGE_UNIT, OTHER THAN $DEFAULT_STOR_UNIT.  IF THERE
-- IS NO OTHER PERMITTED VALUE, THEN USE THE VALUE OF
-- $SYSTEM.STORAGE_UNIT.  IF THERE IS MORE THAN ONE ALTERNATIVE,
-- THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN:  ED7005C1  ED7005D1  CD7005E
NEW_STOR_UNIT        8

-- $DEFAULT_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MEMORY_SIZE.
-- USED IN:  CD7006B  ED7006D3M  CD7006E
DEFAULT_MEM_SIZE     655_360
```

```
-- $NEW_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA MEMORY_SIZE, OTHER THAN $DEFAULT_MEM_SIZE.  IF THERE IS NO
-- OTHER VALUE, THEN USE $DEFAULT_MEM_SIZE.  IF THERE IS MORE THAN
-- ONE ALTERNATIVE, THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR
-- EACH ALTERNATIVE.  IF THE NUMBER OF PERMITTED VALUES IS LARGE, THEN
-- SEVERAL VALUES SHOULD BE USED, COVERING A WIDE RANGE OF
-- POSSIBILITIES.
-- USED IN:  ED7006C1  ED7006D1  CD7006E
NEW_MEM_SIZE          1_048_576

-- $LOW_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE LOWER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN:  CD7007C
LOW_PRIORITY          0

-- $HIGH_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE UPPER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN:  CD7007C
HIGH_PRIORITY         3

-- $MANTISSA_DOC
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_MANTISSA AS SPECIFIED
-- IN THE IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7013B
MANTISSA_DOC          31

-- $DELTA_DOC
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.FINE_DELTA AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7013D
DELTA_DOC             0.000_000_000_465_661_287_307_739_558_602_5

-- $TICK
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.TICK AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7104B
TICK  0.001
```

APPENDIX B

Test Applicability Criteria

Certain tests in the suite may be considered inapplicable to an implementation depending on the way the implementation treats the implementation-dependent features of the language. A brief summary of these implementation-dependent features and the tests they affect are listed in this appendix.

Note that the applicability of each one of these tests is based on the criteria listed in the test file. All potentially inapplicable tests must still be compiled - except those tests which require a floating-point DIGITS value that exceeds SYSTEM.MAX_DIGITS - and executed if successfully compiled, to verify the implementation's treatment of these features. All results for a particular feature must be consistent.

The first part of this appendix is concerned with tests for which the applicability is determined at compile time. Class B tests that are inapplicable should be successfully compiled. Executable test that are inapplicable must be rejected at compile time as a result of the unsupported feature. Lines containing the implementation-dependent features are marked "-- NA => ERROR.".

If the type SHORT_INTEGER is not predefined, then the following tests are inapplicable:

B52004E.DEP	B55B09D.DEP	B86001V.DEP
C45231B.DEP	C45304B.DEP	C45502B.DEP
C45503B.DEP	C45504B.DEP	C45504E.DEP
C45611B.DEP	C45613B.DEP	C45614B.DEP
C45631B.DEP	C45632B.DEP	C55B07B.DEP
CD7101E.DEP		

Compile Time Inapplicability

If the type LONG_INTEGER is not predefined, then the following tests are inapplicable:

B52004D.DEP	B55B09C.DEP	B86001W.DEP
C45231C.DEP	C45304C.DEP	C45502C.DEP
C45503C.DEP	C45504C.DEP	C45504F.DEP
C45611C.DEP	C45613C.DEP	C45614C.DEP
C45631C.DEP	C45632C.DEP	C55B07A.DEP
CD7101F.DEP		

If the type SHORT_FLOAT is not predefined, then the following tests are inapplicable:

B86001T.DEP C35702A.DEP

If the type LONG_FLOAT is not predefined, then the following tests are inapplicable:

B86001U.DEP C35702B.DEP

If there are no predefined integer types other than INTEGER, SHORT_INTEGER, and LONG_INTEGER, then the following tests are inapplicable:

B86001X.TST C45231D.TST CD7101G.TST

If there are no predefined floating-point types other than FLOAT, SHORT_FLOAT, and LONG_FLOAT, then the following test is inapplicable:

B86001Z.TST

If there are no predefined fixed-point types, then the following test is inapplicable:

B86001Y.TST

If 'SIZE representation clauses for boolean types are not supported, then the following tests are inapplicable:

CD2A64A.DEP	CD2A64C.DEP	CD2A65A.DEP
CD2A65C.DEP	CD2A74A.DEP	CD2A76A.DEP

If 'SIZE representation clauses for enumeration types are not supported, then the following tests are inapplicable:

A39005B.DEP	CD1009B.DEP	CD1009P.DEP
CD2A21A.DEP	CD2A21B.DEP	CD2A21C.DEP
CD2A21D.DEP	CD2A21E.DEP	CD2A22A.DEP
CD2A22B.DEP	CD2A22C.DEP	CD2A22D.DEP
CD2A22E.DEP	CD2A22F.DEP	CD2A22G.DEP
CD2A22H.DEP	CD2A22I.DEP	CD2A22J.DEP
CD2A23A.DEP	CD2A23B.DEP	CD2A23C.DEP
CD2A23D.DEP	CD2A23E.DEP	CD2A24A.DEP
CD2A24B.DEP	CD2A24C.DEP	CD2A24D.DEP
CD2A24E.DEP	CD2A24F.DEP	CD2A24G.DEP
CD2A24H.DEP	CD2A24I.DEP	CD2A24J.DEP
CD2A66B.DEP	CD2A66D.DEP	ED2A26A.DEP

If 'SIZE representation clauses for integer types are not supported, then the following tests are inapplicable:

C87B62A.DEP	CD1009A.DEP	CD10090.DEP
CD1C03A.DEP	CD1C04A.DEP	CD2A31A.DEP
CD2A31B.DEP	CD2A31C.DEP	CD2A31D.DEP
CD2A32A.DEP	CD2A32B.DEP	CD2A32C.DEP
CD2A32D.DEP	CD2A32E.DEP	CD2A32F.DEP
CD2A32G.DEP	CD2A32H.DEP	CD2A32I.DEP
CD2A32J.DEP	CD2A64B.DEP	CD2A64D.DEP
CD2A65B.DEP	CD2A65D.DEP	CD2A74B.DEP
CD2A76B.DEP		

If 'SIZE representation clauses for floating-point types are not supported, then the following tests are inapplicable:

CD1009C.DEP	CD2A41A.DEP	CD2A41B.DEP
CD2A41C.DEP	CD2A41D.DEP	CD2A41E.DEP
CD2A42A.DEP	CD2A42B.DEP	CD2A42C.DEP
CD2A42D.DEP	CD2A42E.DEP	CD2A42F.DEP
CD2A42G.DEP	CD2A42H.DEP	CD2A42I.DEP
CD2A42J.DEP		

Compile Time Inapplicability

If 'SIZE representation clauses for fixed-point types are not supported, then the following tests are inapplicable:

CD1009D.DEP	CD1009Q.DEP	CD1C04C.DEP
CD2A51B.DEP	CD2A51C.DEP	CD2A51D.DEP
CD2A51E.DEP	CD2A52A.DEP	CD2A52B.DEP
CD2A52C.DEP	CD2A52D.DEP	CD2A52G.DEP
CD2A52H.DEP	CD2A52I.DEP	CD2A52J.DEP
CD2A53A.DEP	CD2A53B.DEP	CD2A53C.DEP
CD2A53D.DEP	CD2A53E.DEP	CD2A54A.DEP
CD2A54B.DEP	CD2A54C.DEP	CD2A54D.DEP
CD2A54G.DEP	CD2A54H.DEP	CD2A54I.DEP
CD2A54J.DEP	ED2A56A.DEP	

If 'SIZE representation clauses for array types are not supported, then the following tests are inapplicable:

CD1009E.DEP	CD1009F.DEP	CD2A61A.DEP
CD2A61B.DEP	CD2A61C.DEP	CD2A61D.DEP
CD2A61E.DEP	CD2A61F.DEP	CD2A61G.DEP
CD2A61H.DEP	CD2A61I.DEP	CD2A61J.DEP
CD2A61K.DEP	CD2A61L.DEP	CD2A62A.DEP
CD2A62B.DEP	CD2A62C.DEP	CD2A62D.DEP
CD2A63A.DEP	CD2A63B.DEP	CD2A63C.DEP
CD2A63D.DEP	CD2A64A.DEP	CD2A64B.DEP
CD2A64C.DEP	CD2A64D.DEP	CD2A65A.DEP
CD2A65B.DEP	CD2A65C.DEP	CD2A65D.DEP
CD2A66A.DEP	CD2A66B.DEP	CD2A66C.DEP
CD2A66D.DEP		

If 'SIZE representation clauses for record types are not supported, then the following tests are inapplicable:

CD1009G.DEP	CD2A71A.DEP	CD2A71B.DEP
CD2A71C.DEP	CD2A71D.DEP	CD2A72A.DEP
CD2A72B.DEP	CD2A72C.DEP	CD2A72D.DEP
CD2A73A.DEP	CD2A73B.DEP	CD2A73C.DEP
CD2A73D.DEP	CD2A74A.DEP	CD2A74B.DEP
CD2A74C.DEP	CD2A74D.DEP	CD2A75A.DEP
CD2A75B.DEP	CD2A75C.DEP	CD2A75D.DEP
CD2A76A.DEP	CD2A76B.DEP	CD2A76C.DEP
CD2A76D.DEP		

If 'SIZE representation clauses for private types are not supported, then the following test is inapplicable:

CD1009H.DEP

If 'SIZE representation clauses for limited private types are not supported, then the following test is inapplicable:

CD1009I.DEP

If 'SIZE representation clauses for access types are not supported, then the following tests are inapplicable:

CD2A81A.TST	CD2A81B.TST	CD2A81C.TST
CD2A81D.TST	CD2A81E.TST	CD2A81F.TST
CD2A81G.TST	CD2A83A.TST	CD2A83B.TST
CD2A83C.TST	CD2A83E.TST	CD2A83F.TST
CD2A83G.TST	CD2A84A.DEP	CD2A84B.DEP
CD2A84C.DEP	CD2A84D.DEP	CD2A84E.DEP
CD2A84F.DEP	CD2A84G.DEP	CD2A84H.DEP
CD2A84I.DEP	CD2A84K.DEP	CD2A84L.DEP
CD2A84M.DEP	CD2A84N.DEP	CD2A87A.TST
ED2A86A.TST		

If 'SIZE representation clauses for task types are not supported, then the following tests are inapplicable:

CD2A91A.TST	CD2A91B.TST	CD2A91C.TST
CD2A91D.TST	CD2A91E.TST	

If 'STORAGE_SIZE representation clauses for access types are not supported, then the following tests are inapplicable:

A39005C.DEP	C87B62B.DEP	CD1009J.DEP
CD1009R.DEP	CD1009S.DEP	CD1C03C.TST
CD2A83A.TST	CD2A83B.TST	CD2A83C.TST
CD2A83E.TST	CD2A83F.TST	CD2A83G.TST
CD2A84A.DEP	CD2A84B.DEP	CD2A84C.DEP
CD2A84D.DEP	CD2A84E.DEP	CD2A84F.DEP
CD2A84G.DEP	CD2A84H.DEP	CD2A84I.DEP
CD2A84K.DEP	CD2A84L.DEP	CD2A84M.DEP
CD2A84N.DEP	CD2B11A.DEP	CD2B11B.DEP
CD2B11C.DEP	CD2B11D.DEP	CD2B11E.DEP
CD2B11F.DEP	CD2B11G.DEP	CD2B15B.DEP
CD2B15C.DEP	CD2B16A.DEP	CD7205C.DEP
ED2A86A.TST		

If 'STORAGE_SIZE representation clauses for task types are not supported, then the following tests are inapplicable:

A39005D.DEP	C87B62D.DEP	CD1009K.DEP
CD1009T.DEP	CD1009U.DEP	CD1C03E.DEP
CD1C04B.DEP	CD1C06A.DEP	CD7205D.DEP

Compile Time Inapplicability

If the 'SMALL clause is not supported, then the following tests are inapplicable:

A39005E.DEP	C87B62C.DEP	CD1009L.DEP
CD1C03F.DEP	CD2A53A.DEP	CD2A53B.DEP
CD2A53C.DEP	CD2A53D.DEP	CD2A53E.DEP
CD2A54A.DEP	CD2A54B.DEP	CD2A54C.DEP
CD2A54D.DEP	CD2A54G.DEP	CD2A54H.DEP
CD2A54I.DEP	CD2A54J.DEP	CD2D11A.DEP
CD2D11B.DEP	CD2D13A.DEP	ED2A56A.DEP

If enumeration representation clauses are not supported, then the following tests are inapplicable:

AD1009M.DEP	AD1009V.DEP	AD1009W.DEP
AD1C04D.DEP	C35502N.DEP	C35507I.DEP
C35507J.DEP	C35507M.DEP	C35507N.DEP
C35508J.DEP	CD1C03G.DEP	CD2A23A.DEP
CD2A23B.DEP	CD2A23C.DEP	CD2A23D.DEP
CD2A23E.DEP	CD2A24A.DEP	CD2A24B.DEP
CD2A24C.DEP	CD2A24D.DEP	CD2A24E.DEP
CD2A24F.DEP	CD2A24G.DEP	CD2A24H.DEP
CD2A24I.DEP	CD2A24J.DEP	CD3021A.DEP
ED2A26A.DEP		

If enumeration representation clauses for non-contiguous values are not supported, then the following tests are inapplicable:

A39005F.DEP	C35502I.DEP	C35502J.DEP
C35502M.DEP	C35502N.DEP	C35507J.DEP
C35507M.DEP	C35507N.DEP	C35508I.DEP
C35508J.DEP	C35508M.DEP	C35508N.DEP
C55B16A.DEP		

If enumeration representation clauses for types other than character and boolean are not supported, then the following tests are inapplicable:

A39005P.DEP	C35502I.DEP	C35502J.DEP
C35502M.DEP	C35502N.DEP	

If enumeration representation clauses for character types are not supported, then the following tests are inapplicable:

C35507I.DEP	C35507J.DEP	C35507M.DEP
C35507N.DEP	C55B16A.DEP	

If enumeration representation clauses for boolean types, with representation values other than (FALSE => 0, TRUE =>1), are not supported, then the following tests are inapplicable:

C35508I.DEP C35508J.DEP C35508M.DEP
C35508N.DEP

If record representation clauses are not supported, then the following tests are inapplicable:

A39005G.DEP CD1009N.DEP CD1009X.DEP
CD1009Y.DEP CD1009Z.DEP CD1C03H.DEP
CD1C04E.DEP CD4031A.DEP CD4051A.DEP
CD4051B.DEP CD4051C.DEP CD7204C.DEP
ED1D04A.DEP

If 'ADDRESS clauses for variables are not supported, or if the type SYSTEM.ADDRESS is limited, then the following tests are inapplicable:

CD5003B.DEP CD5003C.DEP CD5003D.DEP
CD5003E.DEP CD5003F.DEP CD5003G.DEP
CD5003H.DEP CD5003I.DEP CD5011A.DEP
CD5011C.DEP CD5011E.DEP CD5011G.DEP
CD5011I.DEP CD5011K.DEP CD5011M.DEP
CD5011O.DEP CD5011Q.DEP CD5012A.DEP
CD5012B.DEP CD5012E.DEP CD5012F.DEP
CD5012I.DEP CD5012J.DEP CD5012M.DEP
CD5013A.DEP CD5013C.DEP CD5013E.DEP
CD5013G.DEP CD5013I.DEP CD5013K.DEP
CD5013M.DEP CD5013O.DEP CD5013S.DEP
CD5014A.DEP CD5014C.DEP CD5014E.DEP
CD5014G.DEP CD5014I.DEP CD5014K.DEP
CD5014M.DEP CD5014O.DEP CD5014S.DEP
CD5014T.DEP CD5014V.DEP CD5014X.DEP
CD5014Y.DEP CD5014Z.DEP

Compile Time Inapplicability

If 'ADDRESS clauses for constants are not supported, or if the type SYSTEM.ADDRESS is limited, then the following tests are inapplicable:

CD5011B.DEP	CD5011D.DEP	CD5011F.DEP
CD5011H.DEP	CD5011L.DEP	CD5011N.DEP
CD5011R.DEP	CD5011S.DEP	CD5012C.DEP
CD5012D.DEP	CD5012G.DEP	CD5012H.DEP
CD5012L.DEP	CD5013B.DEP	CD5013D.DEP
CD5013F.DEP	CD5013H.DEP	CD5013J.DEP
CD5013L.DEP	CD5013N.DEP	CD5013R.DEP
CD5014B.DEP	CD5014D.DEP	CD5014F.DEP
CD5014H.DEP	CD5014J.DEP	CD5014L.DEP
CD5014N.DEP	CD5014R.DEP	CD5014U.DEP
CD5014W.DEP		

If 'ADDRESS clauses for subprograms are not supported , or if the type SYSTEM.ADDRESS is limited, then the following tests are inapplicable:

CD5007B.DEP

If the pragmas SYSTEMNAME, STORAGEUNIT, and MEMORYSIZE must come at the beginning of a program library, then the following files are ommitted from the associated test:

ED7004B0.TST	ED7004B2.TST
ED7005C0.TST	ED7005C2.TST
ED7005D0.TST	ED7005D2.TST
ED7006C0.TST	ED7006C2.TST
ED7006D0.TST	ED7006D2.TST

If any limitations, other than those specified in the RM, are imposed on the use of the pragmas SYSTEMNAME, STORAGEUNIT, and MEMORYSIZE, then the following test may be inapplicable:

CD7004C.TST	CD7005E.TST	CD7006E.TST

Consider the following declarations:

```
TYPE F IS DIGITS SYSTEM.MAXDIGITS;
N : CONSTANT := 2.0 * F'MACHINERADIX ** F'MACHINEEMAX;
```

If the declaration of N is rejected on the grounds that evaluation of the expression will raise an exception, then the following test is inapplicable:

C4A013B.ADA

If the fixed-point type definitions

```
DELTA 0.5 RANGE -2.0**10 .. 2.0**10 - 0.5
DELTA 1.0 RANGE -2.0**11 .. 2.0**11 - 1.0
DELTA 2.0**10 RANGE -2.0**21 .. 2.0**21 - 2.0**10
```

are not supported, then the following tests are inapplicable:

```
C45531C.DEP     C45531D.DEP     C45532C.DEP
C45532D.DEP
```

If the fixed-point type definitions

```
DELTA 0.5 RANGE -2.0**14 .. 2.0**14 - 0.5
DELTA 1.0 RANGE -2.0**15 .. 2.0**15 - 1.0
DELTA 2.0**14 RANGE -2.0**29 .. 2.0**29 - 2.0**14
```

are not supported, then the following tests are inapplicable:

```
C45531G.DEP     C45531H.DEP     C45532G.DEP
C45532H.DEP
```

If the fixed-point type definitions

```
DELTA 0.5 RANGE -2.0**30 .. 2.0**30 - 0.5
DELTA 1.0 RANGE -2.0**31 .. 2.0**31 - 1.0
DELTA 2.0**30 RANGE -2.0**61 .. 2.0**61 - 2.0**30
```

are not supported, then the following tests are inapplicable:

```
C45531K.DEP     C45531L.DEP     C45532K.DEP
C45532L.DEP
```

If the fixed-point type definitions

```
DELTA 0.5 RANGE -2.0**46 .. 2.0**46 - 0.5
DELTA 1.0 RANGE -2.0**47 .. 2.0**47 - 1.0
DELTA 2.0**46 RANGE -2.0**93 .. 2.0**93 - 2.0**46
```

are not supported, then the following tests are inapplicable:

```
C455310.DEP     C45531P.DEP     C455320.DEP
C45532P.DEP
```

Compile Time Inapplicability

If the fixed-point type definitions

```
DELTA 2.0**-12 RANGE -0.5 .. 0.5 - 2.0**-12
DELTA 2.0**-11 RANGE -1.0 .. 1.0 - 2.0**-11
DELTA 2.0**-10 RANGE -2.0 .. 2.0 - 2.0**-10
```

are not supported, then the following tests are inapplicable:

C45531A.DEP C45531B.DEP C45532A.DEP
C45532B.DEP

If the fixed-point type definitions

```
DELTA 2.0**-16 RANGE -0.5 .. 0.5 - 2.0**-16
DELTA 2.0**-15 RANGE -1.0 .. 1.0 - 2.0**-15
DELTA 2.0**-14 RANGE -2.0 .. 2.0 - 2.0**-14
```

are not supported, then the following tests are inapplicable:

C45531E.DEP C45531F.DEP C45532E.DEP
C45532F.DEP

If the fixed-point type definitions

```
DELTA 2.0**-32 RANGE -0.5 .. 0.5 - 2.0**-32
DELTA 2.0**-31 RANGE -1.0 .. 1.0 - 2.0**-31
DELTA 2.0**-30 RANGE -2.0 .. 2.0 - 2.0**-30
```

are not supported, then the following tests are inapplicable:

C45531I.DEP C45531J.DEP C45532I.DEP
C45532J.DEP

If the fixed-point type definitions

```
DELTA 2.0**-48 RANGE -0.5 .. 0.5 - 2.0**-48
DELTA 2.0**-47 RANGE -1.0 .. 1.0 - 2.0**-47
DELTA 2.0**-46 RANGE -2.0 .. 2.0 - 2.0**-46
```

are not supported, then the following tests are inapplicable:

C45531M.DEP C45531N.DEP C45532M.DEP
C45532N.DEP

The following tests depend on the value of SYSTEM.MAX_DIGITS:

C24113*.DEP	C35705*.DEP	C35706*.DEP
C35707*.DEP	C35708*.DEP	C35802*.DEP
C45241*.DEP	C45321*.DEP	C45421*.DEP
C45521*.DEP	C45524*.DEP	C45621*.DEP
C45641*.DEP	C46012*.DEP	

Use the table below to determine which of these tests are applicable based on the value of SYSTEM.MAX_DIGITS.

Value of MAX_DIGITS	Last Letter of Test Name of Applicable Tests
5	A
6	A,B
7	A..C
8	A..D
9	A..E
10	A..F
11	A..G
12	A..H
13	A..I
14	A..J
15	A..K
16	A..L
17	A..M
18	A..N
19	A..O
20	A..P
21	A..Q
22	A..R
23	A..S
24	A..T
25	A..U
26	A..V
27	A..W
28	A..X
29	A..Y
30	A..Z

If the value of SYSTEM.MAX_DIGITS > 35, then the following test is inapplicable:

C4A011A.ADA

Compile Time Inapplicability

If a generic library-subprogram body cannot be compiled in a separate file than its specification, then the following test is inapplicable:

CA1012A*.DEP

If a generic library-package body cannot be compiled in a separate file than its specification, then the following tests are inapplicable:

BC3204C*.DEP BC3204D*.DEP

If a generic, non-library package body cannot be compiled as a subunit in a separate file than its specification, then the following test is inapplicable:

CA2009C*.DEP

If a generic, non-library subprogram body cannot be compiled as a subunit in a separate file than its specification, then the following test is inapplicable:

CA2009F*.DEP

If a generic procedure cannot have subunits which are compiled in separate files than the generic unit, then the following test is inapplicable:

CA3011A*.DEP

If package SYSTEM is used by package TEXT_IO, then the following test is inapplicable:

C86001F.DEP

If SEQUENTIAL_IO cannot be instantiated with an unconstrained array type, then the following tests are inapplicable:

AE2101C.DEP EE2201D.ADA

If SEQUENTIAL_IO cannot be instantiated with a record type with discriminants with no default values, then the following tests are inapplicable:

AE2101C.DEP EE2201E.ADA

If DIRECT_IO cannot be instantiated with an unconstrained array type, then the following tests are inapplicable:

AE2101H.DEP EE2401D.ADA

If DIRECT_IO cannot be instantiated with a record type with discriminants with no default values, then the following tests are inapplicable:

AE2101H.DEP EE2401G.ADA

If there are no strings which are illegal as external file names, then the following tests are inapplicable:

CE2102C.TST CE2102H.TST

Tests Which Report "NOT-APPLICABLE"

This section is concerned with tests that can detect, at runtime, certain implementation characteristics that make the tests inapplicable. The result reported for these tests is 'NOT_APPLICABLE'. These tests must compile and execute.

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a text file with mode IN_FILE (this is the appropriate behavior for an implementation which does not support text files other than standard input and output; See AI-00332), then the following test should report NOT_APPLICABLE:

CE3109A.ADA

If CREATE with mode IN_FILE is supported for text files, then the following test should report NOT_APPLICABLE:

CE3102E.ADA

If OPEN with mode IN_FILE is supported for text files, then the following test should report NOT_APPLICABLE:

CE3102J.ADA

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a text file with mode OUT_FILE (this is the appropriate behavior for an implementation which does not support text files other than standard input and output; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2109C.ADA	CE3102A.ADA	CE3102B.TST
CE3107B.ADA	CE3112C.ADA	CE3402D.ADA
CE3403A.ADA	CE3406B.ADA	CE3407B.ADA
CE3408B.ADA	CE3409A.ADA	CE3410A.ADA
CE3605A.ADA	CE3605E.ADA	CE3706F.ADA
CE3905B.ADA	EE3405B.ADA	EE3409F.ADA
EE3410F.ADA		

If CREATE with mode OUT_FILE is supported for text files, then the following test should report NOT_APPLICABLE:

CE3102I.ADA

If OPEN with mode OUT_FILE is supported for text files, then the following test should report NOT_APPLICABLE:

CE3102K.ADA

If RESET is supported for text files, then the following test should report NOT_APPLICABLE:

CE3102F.ADA

If DELETE for text files is supported, then the following test should report NOT_APPLICABLE:

CE3102G.ADA

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a text file (this is the appropriate behavior for an implementation which does not support text files other than standard input and output; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2108A.ADA	CE2108B.ADA	CE2108C.ADA
CE2108D.ADA	CE3102H.ADA	CE3103A.ADA
CE3104A.ADA	CE3104B.ADA	CE3108A.ADA
CE3108B.ADA	CE3110A.ADA	CE3112A.ADA
CE3112B.ADA	CE3112D.ADA	CE3114A.ADA
CE3203A.ADA	CE3208A.ADA	CE3301A.ADA
CE3301B.ADA	CE3301C.ADA	CE3302A.ADA
CE3305A.ADA	CE3402A.ADA	CE3402B.ADA
CE3402C.ADA	CE3403B.ADA	CE3403C.ADA
CE3403E.ADA	CE3403F.ADA	CE3404A.ADA
CE3404B.ADA	CE3404C.ADA	CE3404D.ADA
CE3405A.ADA	CE3405B.ADA	CE3405C.ADA
CE3405D.ADA	CE3406A.ADA	CE3406C.ADA
CE3406D.ADA	CE3407A.ADA	CE3407C.ADA
CE3408A.ADA	CE3408C.ADA	CE3409C.ADA
CE3409D.ADA	CE3409E.ADA	CE3409F.ADA
CE3410C.ADA	CE3410D.ADA	CE3410E.ADA
CE3410F.ADA	CE3411A.ADA	CE3411B.ADA
CE3411C.ADA	CE3412A.ADA	CE3413A.ADA
CE3413C.ADA	CE3602A.ADA	CE3602B.ADA
CE3602C.ADA	CE3602D.ADA	CE3603A.ADA
CE3604A.ADA	CE3604B.ADA	CE3605B.ADA
CE3605C.ADA	CE3605D.ADA	CE3606A.ADA
CE3606B.ADA	CE3704A.ADA	CE3704B.ADA
CE3704D.ADA	CE3704E.ADA	CE3704F.ADA
CE3704M.ADA	CE3704N.ADA	CE3704O.ADA
CE3706D.ADA	CE3706G.ADA	CE3804A.ADA
CE3804B.ADA	CE3804C.ADA	CE3804D.ADA

Tests Which Report "NOT-APPLICABLE"

CE3804E.ADA	CE3804F.ADA	CE3804G.ADA
CE3804H.ADA	CE3804I.ADA	CE3804J.ADA
CE3804K.ADA	CE3804L.ADA	CE3804M.ADA
CE3804N.ADA	CE38040.ADA	CE3804P.ADA
CE3805A.ADA	CE3805B.ADA	CE3806A.ADA
CE3806B.ADA	CE3806D.ADA	CE3806E.ADA
CE3806G.ADA	CE3806H.ADA	CE3905A.ADA
CE3905C.ADA	CE3905L.ADA	CE3906A.ADA
CE3906B.ADA	CE3906C.ADA	CE3906E.ADA
CE3906F.ADA	EE3203A.ADA	EE3402B.ADA
EE3412C.ADA		

If RESET is not supported for text files, then the following tests should report NOT_APPLICABLE:

CE3104C.ADA CE3115A.ADA

If DELETE for text files is not supported, then the following tests should report NOT_APPLICABLE:

CE3110A.ADA CE3114A.ADA

If association of multiple internal text files to a single external file is not supported, then the following tests should report NOT_APPLICABLE:

CE2110B.ADA	CE3111A.ADA	CE3111B.ADA
CE3111C.ADA	CE3111D.ADA	CE3111E.ADA
CE3114B.ADA	CE3115A.ADA	

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a sequential file with mode IN_FILE (this is the appropriate behavior for an implementation which does not support sequential files; see AI-00332)' then the following test should report NOT_APPLICABLE:

CE2105A.ADA

If CREATE and OPEN with mode IN_FILE are supported for sequential files, then the following tests should report NOT_APPLICABLE:

CE2102D.ADA CE2102N.ADA

If RESET to mode IN_FILE is supported for sequential files , then the following tests should report NOT_APPLICABLE:

CE2102O.ADA CE2111F.ADA CE2111I.ADA
CE2204C.ADA

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a sequential file with mode OUT_FILE (this is the appropriate behavior for an implementation which does not support sequential files; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2102A.ADA CE2102C.TST CE2102X.ADA
CE2106A.ADA CE2108E.ADA CE2109A.ADA
CE2110A.ADA

If CREATE and OPEN with mode OUT_FILE are supported, then the following tests should report NOT_APPLICABLE:

CE2102E.ADA CE2102P.ADA

If RESET to mode OUT_FILE is supported for sequential files, then the following test should report NOT_APPLICABLE:

CE2102Q.ADA

If RESET to mode OUT_FILE is not supported for sequential files, then the following tests should report NOT_APPLICABLE:

CE2111C.ADA CE2111D.ADA CE2111F.ADA
CE2111I.ADA

If DELETE for sequential files is not supported, then the following tests should report NOT_APPLICABLE:

CE2106A.ADA CE2110A.ADA

If association of multiple internal sequential files to a single external file is not supported, then the following tests should report NOT_APPLICABLE:

CE2107A.ADA CE2107B.ADA CE2107C.ADA
CE2107D.ADA CE2111D.ADA

Tests Which Report "NOT-APPLICABLE"

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a sequential file (this is the appropriate behavior for an implementation which does not support sequential files; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2103C.ADA	CE2102G.ADA	CE2104A.ADA
CE2104B.ADA	CE2108A.ADA	CE2108B.ADA
CE2108F.ADA	CE2111A.ADA	CE2115B.ADA
CE2201A.ADA	CE2201B.ADA	CE2201C.ADA
CE2201F.ADA	CE2201G.ADA	CE2201H.ADA
CE2201I.ADA	CE2201J.ADA	CE2201K.ADA
CE2201L.ADA	CE2201M.ADA	CE2201N.ADA
CE2204A.ADA	CE2204B.ADA	CE2204C.ADA
CE2204D.ADA	CE2205A.ADA	CE2208B.ADA
CE2210A.ADA		

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a direct access file with mode IN_FILE (this is the appropriate behavior for an implementation which does not support direct access files; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2105B.ADA	CE2401E.ADA	CE2401F.ADA
CE2401H.ADA	CE2401I.ADA	CE2401J.ADA
CE2405B.ADA	CE2407A.ADA	

If CREATE with mode IN_FILE is supported for direct access files, then the following test should report NOT_APPLICABLE:

CE2102I.ADA

If OPEN with mode IN_FILE is supported for direct access files, then the following test should report NOT_APPLICABLE:

CE2102T.ADA

If RESET with mode IN_FILE is supported for direct access files, then the following tests should report NOT_APPLICABLE:

CE2102U.ADA	CE2407B.ADA

If RESET with mode IN_FILE is not supported for direct access files, then the following tests should report NOT_APPLICABLE:

CE2111B.ADA	CE2111G.ADA	CE2111H.ADA

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a direct access file with mode OUT_FILE (this is the appropriate behavior for an implementation which does not support direct access files; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2102B.ADA	CE2102Y.ADA	CE2106B.ADA
CE2110C.ADA	CE2401K.ADA	CE2404A.ADA
CE2404B.ADA	CE2407A.ADA	CE2407B.ADA
CE2408A.ADA	CE2409B.ADA	CE2410A.ADA
CE2410B.ADA		

If CREATE and OPEN with mode OUT_FILE is supported for direct access files, then the following tests should report NOT_APPLICABLE:

CE2102J.ADA	CE2102V.ADA

If RESET with mode OUT_FILE is supported for direct access files, then the following test should report NOT_APPLICABLE:

CE2102W.ADA

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a direct access file with mode INOUT_FILE (this is the appropriate behavior for an implementation which does not support direct access files; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2102H.TST	CE2109B.ADA	CE2401E.ADA
CE2401F.ADA	CE2401H.ADA	CE2401I.ADA
CE2401J.ADA	CE2401K.ADA	CE2401L.ADA
CE2405B.ADA	CE2408B.ADA	CE2409A.ADA

If CREATE and OPEN with mode INOUT_FILE are supported for direct access files, then the following tests should report NOT_APPLICABLE:

CE2102F.ADA	CE2102R.ADA

If RESET to mode INOUT_FILE is supported for direct access files, then the following test should report NOT_APPLICABLE:

CE2102S.ADA

Tests Which Report "NOT-APPLICABLE"

If DELETE for direct access files is not supported, then the following tests should report NOT_APPLICABLE:

CE2106B.ADA CE2110C.ADA

If association of multiple internal direct access files to a single external file is not supported, then the following tests should report NOT_APPLICABLE:

CE2107F.ADA CE2107G.ADA CE2107H.ADA
CE2107I.ADA CE2110D.ADA CE2111H.ADA

If USE_ERROR or NAME_ERROR is raised by every attempt to create or open a direct access file (this is the appropriate behavior for an implementation which does not support direct access files; see AI-00332), then the following tests should report NOT_APPLICABLE:

CE2102K.ADA CE2103D.ADA CE2104C.ADA
CE2104D.ADA CE2108C.ADA CE2108D.ADA
CE2108G.ADA CE2108H.ADA CE2111E.ADA
CE2115A.ADA CE2401A.ADA CE2401B.ADA
CE2401C.ADA CE2406A.ADA CE2411A.ADA

If association of an internal sequential file and an internal direct access file to a single external file is not supported, then the following tests should report NOT_APPLICABLE:

CE2107E.ADA CE2107L.ADA

If the INLINE pragma is not supported for PROCEDURES, then the following tests should report NOT_APPLICABLE:

CA3004E*.ADA EA3004C*.ADA LA3004A*.ADA

If the INLINE pragma is not supported for FUNCTIONS, then the following tests should report NOT_APPLICABLE:

CA3004F*.ADA EA3004D*.ADA LA3004B*.ADA

If excessive recursion causes STORAGE_ERROR to be raised, then the following test should report NOT_APPLICABLE:

D64005F*.ADA

If DURATION'FIRST = DURATION'BASE'FIRST or DURATION'LAST = DURATION'BASE'LAST, then the following test should report NOT_APPLICABLE, if it otherwise executes properly:

C96005B.TST

Consider the declaration

```
TYPE F IS DIGITS SYSTEM.MAX_DIGITS;
```

If F'MACHINE_OVERFLOWS is false and 2.0 * F'MACHINE_RADIX ** F'MACHINE_EMAX <= F'BASE'LARGE, then the following test should report NOT_APPLICABLE if it compiles without error:

C4A013B.ADA

APPENDIX C

Package REPORT Specification and Body

```
-- REPSPEC.ADA

-- PURPOSE:
--      THIS REPORT PACKAGE PROVIDES THE MECHANISM FOR REPORTING THE
--      PASS/FAIL/NOT-APPLICABLE RESULTS OF EXECUTABLE (CLASSES A, C,
--      D, E, AND L) TESTS.

--      IT ALSO PROVIDES THE MECHANISM FOR GUARANTEEING THAT CERTAIN
--      VALUES BECOME DYNAMIC (NOT KNOWN AT COMPILE-TIME).

-- HISTORY:
--      JRK 12/13/79
--      JRK 06/10/80
--      JRK 08/06/81
--      JRK 10/27/82
--      JRK 06/01/84
--      PWB 07/30/87  ADDED PROCEDURE SPECIAL_ACTION.
--      TBN 08/20/87  ADDED FUNCTION LEGAL_FILE_NAME.

PACKAGE REPORT IS

     SUBTYPE FILE_NUM IS INTEGER RANGE 1..3;

  -- THE REPORT ROUTINES.

     PROCEDURE TEST              -- THIS ROUTINE MUST BE INVOKED AT THE
                                 -- START OF A TEST, BEFORE ANY OF THE
                                 -- OTHER REPORT ROUTINES ARE INVOKED.
                                 -- IT SAVES THE TEST NAME AND OUTPUTS THE
                                 -- NAME AND DESCRIPTION.
        ( NAME : STRING;         -- TEST NAME, E.G., "C23001A-AB".
          DESCR : STRING         -- BRIEF DESCRIPTION OF TEST, E.G.,
                                 -- "UPPER/LOWER CASE EQUIVALENCE IN " &
```

```
                                -- "IDENTIFIERS".
      );

   PROCEDURE FAILED             -- OUTPUT A FAILURE MESSAGE.  SHOULD BE
                                -- INVOKED SEPARATELY TO REPORT THE
                                -- FAILURE OF EACH SUBTEST WITHIN A TEST.
      ( DESCR : STRING          -- BRIEF DESCRIPTION OF WHAT FAILED.
                                -- SHOULD BE PHRASED AS:
                                -- "(FAILED BECAUSE) ...REASON...".
      );

   PROCEDURE NOT_APPLICABLE     -- OUTPUT A NOT-APPLICABLE MESSAGE.
                                -- SHOULD BE INVOKED SEPARATELY TO REPORT
                                -- THE NON-APPLICABILITY OF EACH SUBTEST
                                -- WITHIN A TEST.
      ( DESCR : STRING          -- BRIEF DESCRIPTION OF WHAT IS
                                -- NOT-APPLICABLE. SHOULD BE PHRASED AS:
                                -- "(NOT-APPLICABLE BECAUSE)...REASON...".
      );

   PROCEDURE SPECIAL_ACTION     -- OUTPUT A MESSAGE DESCRIBING SPECIAL
                                -- ACTIONS TO BE TAKEN.
                                -- SHOULD BE INVOKED SEPARATELY TO GIVE
                                -- EACH SPECIAL ACTION.
      ( DESCR : STRING          -- BRIEF DESCRIPTION OF ACTION TO BE
                                -- TAKEN.
      );

   PROCEDURE COMMENT            -- OUTPUT A COMMENT MESSAGE.
      ( DESCR : STRING          -- THE MESSAGE.
      );

   PROCEDURE RESULT;            -- THIS ROUTINE MUST BE INVOKED AT THE
                                -- END OF A TEST.  IT OUTPUTS A MESSAGE
                                -- INDICATING WHETHER THE TEST AS A
                                -- WHOLE HAS PASSED, FAILED, IS
                                -- NOT-APPLICABLE, OR HAS TENTATIVELY
                                -- PASSED PENDING SPECIAL ACTIONS.

-- THE DYNAMIC VALUE ROUTINES.

   -- EVEN WITH STATIC ARGUMENTS, THESE FUNCTIONS WILL HAVE DYNAMIC
   -- RESULTS.

   FUNCTION IDENT_INT           -- AN IDENTITY FUNCTION FOR TYPE INTEGER.
      ( X : INTEGER             -- THE ARGUMENT.
      ) RETURN INTEGER;         -- X.

   FUNCTION IDENT_CHAR          -- AN IDENTITY FUNCTION FOR TYPE
                                -- CHARACTER.
      ( X : CHARACTER           -- THE ARGUMENT.
      ) RETURN CHARACTER;       -- X.
```

```
     FUNCTION IDENT_BOOL       -- AN IDENTITY FUNCTION FOR TYPE BOOLEAN.
        ( X : BOOLEAN          -- THE ARGUMENT.
        ) RETURN BOOLEAN;      -- X.

     FUNCTION IDENT_STR        -- AN IDENTITY FUNCTION FOR TYPE STRING.
        ( X : STRING           -- THE ARGUMENT.
        ) RETURN STRING;       -- X.

     FUNCTION EQUAL            -- A RECURSIVE EQUALITY FUNCTION FOR TYPE
                               -- INTEGER.
        ( X, Y : INTEGER       -- THE ARGUMENTS.
        ) RETURN BOOLEAN;      -- X = Y.

-- OTHER UTILITY ROUTINES.

     FUNCTION LEGAL_FILE_NAME -- A FUNCTION TO GENERATE LEGAL EXTERNAL
                               -- FILE NAMES.
        ( X : FILE_NUM := 1;   -- DETERMINES FIRST CHARACTER OF NAME.
          NAM : STRING := ""   -- DETERMINES REST OF NAME.
        ) RETURN STRING;       -- THE GENERATED NAME.

END REPORT;
```

REPORT Package Body

```
-- REPBODY.ADA

-- HISTORY:
--      DCB 04/27/80
--      JRK 6/10/80
--      JRK 11/12/80
--      JRK 8/6/81
--      JRK 10/27/82
--      JRK 6/1/84
--      JRK 11/18/85  ADDED PRAGMA ELABORATE.
--      PWB 07/29/87  ADDED STATUS ACTION_REQUIRED AND
--                    PROCEDURE SPECIAL_ACTION.
--      TBN 08/20/87  ADDED FUNCTION LEGAL_FILE_NAME.

WITH TEXT_IO;
USE TEXT_IO;
PRAGMA ELABORATE (TEXT_IO);

PACKAGE BODY REPORT IS

     TYPE STATUS IS (PASS, FAIL, DOES_NOT_APPLY, ACTION_REQUIRED);
     TEST_STATUS : STATUS := FAIL;

     MAX_NAME_LEN : CONSTANT := 15;      -- MAXIMUM TEST NAME LENGTH.
     TEST_NAME : STRING (1..MAX_NAME_LEN);

     NO_NAME : CONSTANT STRING (1..7) := "NO NAME";
     TEST_NAME_LEN : INTEGER RANGE 0..MAX_NAME_LEN := 0;

     PROCEDURE PUT_MSG (MSG : STRING) IS
          -- WRITE MESSAGE.  LONG MESSAGES ARE FOLDED (AND INDENTED).
          MAX_LEN : CONSTANT INTEGER RANGE 50..150 := 72;  -- MAXIMUM
                                          -- OUTPUT LINE LENGTH.
          INDENT : CONSTANT INTEGER := TEST_NAME_LEN + 9;  -- AMOUNT TO
                                          -- INDENT CONTINUATION LINES.
          I : INTEGER := 0;               -- CURRENT INDENTATION.
          M : INTEGER := MSG'FIRST;       -- START OF MESSAGE SLICE.
          N : INTEGER;                    -- END OF MESSAGE SLICE.
     BEGIN
          LOOP
               IF I + (MSG'LAST-M+1) > MAX_LEN THEN
                    N := M + (MAX_LEN-I) - 1;
                    IF MSG (N) /= ' ' THEN
                         WHILE N >= M AND THEN MSG (N+1) /= ' ' LOOP
                              N := N - 1;
                         END LOOP;
                         IF N < M THEN
                              N := M + (MAX_LEN-I) - 1;
                         END IF;
                    END IF;
               ELSE N := MSG'LAST;
               END IF;
```

```
            SET_COL (STANDARD_OUTPUT, COUNT (I+1));
            PUT_LINE (STANDARD_OUTPUT, MSG (M..N));
            I := INDENT;
            M := N + 1;
            WHILE M <= MSG'LAST AND THEN MSG (M) = ' ' LOOP
                 M := M + 1;
            END LOOP;
            EXIT WHEN M > MSG'LAST;
       END LOOP;
  END PUT_MSG;

  PROCEDURE TEST (NAME : STRING; DESCR : STRING) IS
  BEGIN
       TEST_STATUS := PASS;
       IF NAME'LENGTH <= MAX_NAME_LEN THEN
            TEST_NAME_LEN := NAME'LENGTH;
       ELSE TEST_NAME_LEN := MAX_NAME_LEN;
       END IF;
       TEST_NAME (1..TEST_NAME_LEN) :=
                 NAME (NAME'FIRST .. NAME'FIRST+TEST_NAME_LEN-1);
       PUT_MSG ("");
       PUT_MSG ("---- " & TEST_NAME (1..TEST_NAME_LEN) & " " &
                 DESCR & ".");
  END TEST;

  PROCEDURE COMMENT (DESCR : STRING) IS
  BEGIN
       PUT_MSG ("    - " & TEST_NAME (1..TEST_NAME_LEN) & " " &
                 DESCR & ".");
  END COMMENT;

  PROCEDURE FAILED (DESCR : STRING) IS
  BEGIN
       TEST_STATUS := FAIL;
       PUT_MSG ("    * " & TEST_NAME (1..TEST_NAME_LEN) & " " &
                 DESCR & ".");
  END FAILED;

  PROCEDURE NOT_APPLICABLE (DESCR : STRING) IS
  BEGIN
       IF TEST_STATUS = PASS OR TEST_STATUS = ACTION_REQUIRED THEN
            TEST_STATUS := DOES_NOT_APPLY;
       END IF;
       PUT_MSG ("    + " & TEST_NAME (1..TEST_NAME_LEN) & " " &
                 DESCR & ".");
  END NOT_APPLICABLE;

  PROCEDURE SPECIAL_ACTION (DESCR : STRING) IS
  BEGIN
       IF TEST_STATUS = PASS THEN
            TEST_STATUS := ACTION_REQUIRED;
       END IF;
```

```
          PUT_MSG ("    ! " & TEST_NAME (1..TEST_NAME_LEN) & " " &
                   DESCR & ".");
     END SPECIAL_ACTION;

     PROCEDURE RESULT IS
     BEGIN
          CASE TEST_STATUS IS
          WHEN PASS =>
               PUT_MSG ("==== " & TEST_NAME (1..TEST_NAME_LEN) &
                        " PASSED ==========================.");
          WHEN DOES_NOT_APPLY =>
               PUT_MSG ("++++ " & TEST_NAME (1..TEST_NAME_LEN) &
                        " NOT-APPLICABLE ++++++++++++++++++++.");
          WHEN ACTION_REQUIRED =>
               PUT_MSG ("!!!! " & TEST_NAME (1..TEST_NAME_LEN) &
                        " TENTATIVELY PASSED !!!!!!!!!!!!!!!!.");
               PUT_MSG ("!!!! " & (1..TEST_NAME_LEN => ' ') &
                        " SEE '!' COMMENTS FOR SPECIAL NOTES!!");
          WHEN OTHERS =>
               PUT_MSG ("**** " & TEST_NAME (1..TEST_NAME_LEN) &
                        " FAILED **************************.");
          END CASE;
          TEST_STATUS := FAIL;
          TEST_NAME_LEN := NO_NAME'LENGTH;
          TEST_NAME (1..TEST_NAME_LEN) := NO_NAME;
     END RESULT;

FUNCTION IDENT_INT (X : INTEGER) RETURN INTEGER IS
     BEGIN
          IF EQUAL (X, X) THEN          -- ALWAYS EQUAL.
               RETURN X;                -- ALWAYS EXECUTED.
          END IF;
          RETURN 0;                     -- NEVER EXECUTED.
     END IDENT_INT;

     FUNCTION IDENT_CHAR (X : CHARACTER) RETURN CHARACTER IS
     BEGIN
          IF EQUAL (CHARACTER'POS(X), CHARACTER'POS(X)) THEN  -- ALWAYS
                                        -- EQUAL.
               RETURN X;                -- ALWAYS EXECUTED.
          END IF;
          RETURN '0';                   -- NEVER EXECUTED.
     END IDENT_CHAR;

     FUNCTION IDENT_BOOL (X : BOOLEAN) RETURN BOOLEAN IS
     BEGIN
          IF EQUAL (BOOLEAN'POS(X), BOOLEAN'POS(X)) THEN  -- ALWAYS
                                        -- EQUAL.
               RETURN X;                -- ALWAYS EXECUTED.
          END IF;
          RETURN FALSE;                 -- NEVER EXECUTED.
     END IDENT_BOOL;
```

```
     FUNCTION IDENT_STR (X : STRING) RETURN STRING IS
     BEGIN
          IF EQUAL (X'LENGTH, X'LENGTH) THEN  -- ALWAYS EQUAL.
               RETURN X;                      -- ALWAYS EXECUTED.
          END IF;
          RETURN "";                          -- NEVER EXECUTED.
     END IDENT_STR;

     FUNCTION EQUAL (X, Y : INTEGER) RETURN BOOLEAN IS
          REC_LIMIT : CONSTANT INTEGER RANGE 1..100 := 3;  -- RECURSION
                                              -- LIMIT.
          Z : BOOLEAN;                        -- RESULT.
     BEGIN
          IF X < 0 THEN
               IF Y < 0 THEN
                    Z := EQUAL (-X, -Y);
               ELSE Z := FALSE;
               END IF;
          ELSIF X > REC_LIMIT THEN
               Z := EQUAL (REC_LIMIT, Y-X+REC_LIMIT);
          ELSIF X > 0 THEN
               Z := EQUAL (X-1, Y-1);
          ELSE Z := Y = 0;
          END IF;
          RETURN Z;
     EXCEPTION
          WHEN OTHERS =>
               RETURN X = Y;
     END EQUAL;

     FUNCTION LEGAL_FILE_NAME (X : FILE_NUM := 1;
                               NAM : STRING := "")
                              RETURN STRING IS
          SUFFIX : STRING (2..6);
     BEGIN
          IF NAM = "" THEN
               SUFFIX := TEST_NAME(3..7);
          ELSE
               SUFFIX := NAM(3..7);
          END IF;

          CASE X IS
               WHEN 1 => RETURN ('X' & SUFFIX);
               WHEN 2 => RETURN ('Y' & SUFFIX);
               WHEN 3 => RETURN ('Z' & SUFFIX);
          END CASE;
     END LEGAL_FILE_NAME;

BEGIN

     TEST_NAME_LEN := NO_NAME'LENGTH;
     TEST_NAME (1..TEST_NAME_LEN) := NO_NAME;
```

```
REPORT Package Body

END REPORT;
```

APPENDIX D

Procedure CHECKFILE Body

```
-- CHECK_FILE.ADA

-- THIS IS A PROCEDURE USED BY MANY OF THE CHAPTER 14 TESTS TO CHECK THE
-- CONTENTS OF A TEXT FILE.

-- THIS PROCEDURE ASSUMES THE FILE PARAMETER PASSED TO IT IS AN OPEN
-- TEXT FILE.

-- THE STRING PARAMETER CONTAINS THE CHARACTERS THAT ARE SUPPOSED TO BE
-- IN THE TEXT FILE.  A '#' CHARACTER IS USED IN THE STRING TO DENOTE
-- THE END OF A LINE.  A '@' CHARACTER IS USED TO DENOTE THE END OF A
-- PAGE. A '%' CHARACTER IS USED TO DENOTE THE END OF THE TEXT FILE.
-- THESE SYMBOLS SHOULD NOT BE USED AS TEXT OUTPUT.

-- SPS 11/30/82
-- JBG 2/3/83

WITH REPORT; USE REPORT;
WITH TEXT_IO; USE TEXT_IO;

PROCEDURE CHECK_FILE (FILE: IN OUT FILE_TYPE; CONTENTS : STRING) IS

     X : CHARACTER;
     COL_COUNT : POSITIVE_COUNT := 1;
     LINE_COUNT : POSITIVE_COUNT := 1;
     PAGE_COUNT : POSITIVE_COUNT := 1;
     TRAILING_BLANKS_MSG_WRITTEN : BOOLEAN := FALSE;
     STOP_PROCESSING : EXCEPTION;

     PROCEDURE CHECK_END_OF_LINE (EXPECT_END_OF_PAGE : BOOLEAN) IS
     BEGIN

-- SKIP OVER ANY TRAILING BLANKS.  AN IMPLEMENTATION CAN LEGALLY
```

```
Procedure CHECKFILE Body


-- APPEND BLANKS TO THE END OF ANY LINE.

          WHILE NOT END_OF_LINE (FILE) LOOP
               GET (FILE, X);
               IF X /= ' ' THEN
                    FAILED ("FROM CHECK_FILE: END OF LINE EXPECTED - " &
                            X & " ENCOUNTERED");
                    RAISE STOP_PROCESSING;
               ELSE
                    IF NOT TRAILING_BLANKS_MSG_WRITTEN THEN
                         COMMENT ("FROM CHECK_FILE: " &
                                  "THIS IMPLEMENTATION PADS " &
                                  "LINES WITH BLANKS");
                         TRAILING_BLANKS_MSG_WRITTEN := TRUE;
                    END IF;
               END IF;
          END LOOP;

          IF LINE_COUNT /= LINE (FILE) THEN
               FAILED ("FROM CHECK_FILE: " &
                       "LINE COUNT INCORRECT - EXPECTED " &
                       POSITIVE_COUNT'IMAGE(LINE_COUNT) &
                       " GOT FROM FILE " &
                       POSITIVE_COUNT'IMAGE(LINE(FILE)));
          END IF;

-- NOTE:  DO NOT SKIP_LINE WHEN AT END OF PAGE BECAUSE SKIP_LINE WILL
--        ALSO SKIP THE PAGE TERMINATOR.  SEE RM 14.3.5 PARAGRAPH 1.

          IF NOT EXPECT_END_OF_PAGE THEN
               IF END_OF_PAGE (FILE) THEN
                    FAILED ("FROM CHECK_FILE: PREMATURE END OF PAGE");
                    RAISE STOP_PROCESSING;
               ELSE
                    SKIP_LINE (FILE);
                    LINE_COUNT := LINE_COUNT + 1;
               END IF;
          END IF;
          COL_COUNT := 1;
     END CHECK_END_OF_LINE;

     PROCEDURE CHECK_END_OF_PAGE IS
     BEGIN
          IF NOT END_OF_PAGE (FILE) THEN
               FAILED ("FROM CHECK_FILE: " &
                       "END_OF_PAGE NOT WHERE EXPECTED");
               RAISE STOP_PROCESSING;
          ELSE
               IF PAGE_COUNT /= PAGE (FILE) THEN
                    FAILED ("FROM CHECK_FILE: " &
                            "PAGE COUNT INCORRECT - EXPECTED " &
                            POSITIVE_COUNT'IMAGE (PAGE_COUNT) &
```

```
                                    " GOT FROM FILE " &
                                    POSITIVE_COUNT'IMAGE (PAGE(FILE)));
                    END IF;

                    SKIP_PAGE (FILE);
                    PAGE_COUNT := PAGE_COUNT + 1;
                    LINE_COUNT := 1;
               END IF;
          END CHECK_END_OF_PAGE;

     BEGIN

          RESET (FILE, IN_FILE);
          SET_LINE_LENGTH (STANDARD_OUTPUT, 0);
          SET_PAGE_LENGTH (STANDARD_OUTPUT, 0);

          FOR I IN 1 .. CONTENTS'LENGTH LOOP

               BEGIN
                    CASE CONTENTS (I) IS
                         WHEN '#' =>
                              CHECK_END_OF_LINE (CONTENTS (I + 1) = '@');
                         WHEN '@' =>
                              CHECK_END_OF_PAGE;
                         WHEN '%' =>
                              IF NOT END_OF_FILE (FILE) THEN
                                   FAILED ("FROM CHECK_FILE: " &
                                           "END_OF_FILE NOT WHERE EXPECTED");
                                   RAISE STOP_PROCESSING;
                              END IF;
                         WHEN OTHERS =>
                              IF COL_COUNT /= COL(FILE) THEN
                                   FAILED ("FROM CHECK_FILE: " &
                                           "COL COUNT INCORRECT - " &
                                           "EXPECTED " & POSITIVE_COUNT'
                                           IMAGE(COL_COUNT) & " GOT FROM " &
                                           "FILE " & POSITIVE_COUNT'IMAGE
                                           (COL(FILE)));
                              END IF;
                              GET (FILE, X);
                              COL_COUNT := COL_COUNT + 1;
                              IF X /= CONTENTS (I) THEN
                                   FAILED("FROM CHECK_FILE: " &
                                          "FILE DOES NOT CONTAIN CORRECT " &
                                          "OUTPUT - EXPECTED " & CONTENTS(I)
                                          & " - GOT " & X);
                                   RAISE STOP_PROCESSING;
                              END IF;
                    END CASE;
               EXCEPTION
                    WHEN STOP_PROCESSING =>
                         COMMENT ("FROM CHECK_FILE: " &
```

```
Procedure CHECKFILE Body

                                "LAST CHARACTER IN FOLLOWING STRING " &
                                "REVEALED ERROR: " & CONTENTS (1 .. I));
                        EXIT;
                END;

        END LOOP;

EXCEPTION
        WHEN STATUS_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "STATUS_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN MODE_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "MODE_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN NAME_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "NAME_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN USE_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "USE_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN DEVICE_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "DEVICE_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN END_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "END_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN DATA_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "DATA_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN LAYOUT_ERROR =>
                FAILED ("FROM CHECK_FILE: " &
                        "LAYOUT_ERROR RAISED - FILE CHECKING INCOMPLETE");
        WHEN OTHERS =>
                FAILED ("FROM CHECK_FILE: " &
                        "SOME EXCEPTION RAISED - FILE CHECKING INCOMPLETE");

END CHECK_FILE;
```

APPENDIX E

Sample Output Of Acceptance Tests

Sample Output Of Acceptance Tests

Test CZ1101A.ADA should produce the following on STANDARD_OUTPUT:

```
   - NO_NAME CZ1101A-AB: CHECK REPORT ROUTINES. CHECKING 'COMMENT'.
   - NO_NAME CHECKING 'PUT_MSG' FOR PROPER OUTPUT.
   - NO_NAME THIS LINE IS EXACTLY 'MAX_LEN' LONG. ....5...60....5...70..
   - NO_NAME THIS COMMENT HAS A WORD THAT SPANS THE FOLD POINT. THIS
               COMMENT FITS EXACTLY ON TWO LINES. ...5...60....5...70..
   - NO_NAME THIS COMMENT HAS A WORD THAT ENDS AT THE FOLD POINT ...70..
               THE SPACES JUST AFTER THE FOLD POINT SHOULD BE GONE.
   - NO_NAME THIS COMMENT HAS A WORD THAT BEGINS AT THE FOLD POINT.
               THIS SHOULD BE ON THE SECOND LINE.
   - NO_NAME THIS COMMENT HAS SPACES SPANNING THE FOLD POINT. ..5...70
               THIS SHOULD BE ON THE SECOND LINE.
   - NO_NAME
               THIS_COMMENT_IS_ONE_VERY_LONG_WORD_AND_SO_IT_SHOULD_BE_S
               PLIT_AT_THE_FOLD_POINT.
   - NO_NAME CHECKING THAT REPORT BODY INITIALIZES TO 'NO_NAME' AND
               'FAILED'. CHECKING 'RESULT'.
**** NO_NAME FAILED ********************.

---- PASS_TEST CHECKING 'TEST' AND 'RESULT' FOR 'PASSED'.
   - PASS_TEST CHECKING THAT INDENTATION OF CONTINUATION LINES IS
                 ADJUSTED ACCORDING TO TEST NAME LENGTH.
==== PASS_TEST PASSED ====================.
   - NO_NAME CHECKING 'RESULT' FOR RESETTING TO 'NO_NAME' AND 'FAILED'.
**** NO_NAME FAILED ********************.
   - NO_NAME CHECKING THAT INDENTATION OF CONTINUATION LINES IS ADJUSTED
               ACCORDING TO TEST NAME LENGTH.

---- FAIL_TEST CHECKING 'FAILED' AND 'RESULT' FOR 'FAILED'.
   * FAIL_TEST CHECKING 'FAILED'. NOW 'RESULT' SHOULD BE 'FAILED'.
**** FAIL_TEST FAILED ********************.

---- NA_TEST CHECKING 'NOT_APPLICABLE' AND 'RESULT' FOR
               'NOT-APPLICABLE'.
   + NA_TEST CHECKING 'NOT_APPLICABLE'. NOW 'RESULT' SHOULD BE
               'NOT-APPLICABLE'.
++++ NA_TEST NOT-APPLICABLE ++++++++++++++++++++.

---- FAIL_NA_TEST CHECKING 'FAILED', 'NOT_APPLICABLE', AND 'RESULT' FOR
                   'FAILED'.
   + FAIL_NA_TEST CHECKING 'NOT_APPLICABLE'. NOW 'RESULT' SHOULD BE
                   'NOT-APPLICABLE'.
   * FAIL_NA_TEST CHECKING 'FAILED'. NOW 'RESULT' SHOULD BE 'FAILED'.
   + FAIL_NA_TEST CHECKING 'NOT_APPLICABLE'. NOW 'RESULT' SHOULD STAY
                   'FAILED'.
**** FAIL_NA_TEST FAILED ********************.
   - NO_NAME END CZ1101A.
```

Test CZ1103A.ADA should produce the following on STANDARD_OUTPUT:

```
---- CZ1103A CHECK THAT PROCEDURE CHECK_FILE WORKS.
   - CZ1103A BEGIN TEST WITH AN EMPTY FILE.
   - CZ1103A BEGIN TEST WITH A FILE WITH BLANK LINES.
   - CZ1103A BEGIN TEST WITH A FILE WITH BLANK LINES AND PAGES.
   - CZ1103A BEGIN TEST WITH A FILE WITH TRAILING BLANKS.
   - CZ1103A FROM CHECK_FILE: THIS IMPLEMENTATION PADS LINES WITH
               BLANKS.
   - CZ1103A BEGIN TEST WITH A FILE WITHOUT TRAILING BLANKS.
   - CZ1103A BEGIN TEST WITH A FILE WITH AN END OF LINE ERROR.
   * CZ1103A FROM CHECK_FILE: END OF LINE EXPECTED - E ENCOUNTERED.
   - CZ1103A FROM CHECK_FILE: LAST CHARACTER IN FOLLOWING STRING
               REVEALED ERROR: THIS LINE WILL CONTAIN AN #.
   - CZ1103A BEGIN TEST WITH FILE WITH END OF PAGE ERROR.
   * CZ1103A FROM CHECK_FILE: END OF PAGE NOT WHERE EXPECTED.
   - CZ1103A FROM CHECK_FILE: LAST CHARACTER IN FOLLOWING STRING
               REVEALED ERROR: THIS LINE WILL CONTAIN AN @.
   - CZ1103A BEGIN TEST WITH FILE WITH END OF FILE ERROR.
   * CZ1103A FROM CHECK_FILE: END OF FILE NOT WHERE EXPECTED.
   - CZ1103A FROM CHECK_FILE: LAST CHARACTER IN FOLLOWING STRING
               REVEALED ERROR: THIS LINE WILL CONTAIN AN %.
   - CZ1103A BEGIN TEST WITH FILE WITH INCORRECT DATA.
   * CZ1103A FROM CHECK_FILE: FILE DOES NOT CONTAIN CORRECT OUTPUT -
               EXPECTED C - GOT I.
   - CZ1103A FROM CHECK_FILE: LAST CHARACTER IN FOLLOWING STRING
               REVEALED ERROR: LINE WITH C.
**** CZ1103A FAILED ********************.
```